PROGRAMMABLE LOGIC CONTROLLER FOR INTERPRETING DECISION TABLES

A Thesis Submitted
in partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
RAVINDRA PRAKASH

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

JULY 1975

CERTIFICATE

This is to certify that the thesis entitled "Programmable Logic Controller for Interpreting Decision Tables" by Ravindra Prakash, is a record of work carried out under my supervision and has not been submitted elsewhere for a degree.

[illegible]

July - 1975

[V. RAJARAMAN]
PROFESSOR
COMPUTER CENTRE
INDIAN INSTITUTE OF TECHNOLOGY
KANPUR-208016

EE-1975-M-PRA-PRO

ACKNOWLEDGEMENT

I am deeply indebted to Dr. V. Rajaraman for his continued guidance, supervision, and encouragement throughout this work.

I would like to express my gratitude to Engineers India Limited, Delhi who generously supplied the relay diagrams used in industries.

I am also very grateful to Dr. K.L. Kodandapani, Messrs. M.S. Krishnamoorthy, Anil K. Agarwal, Kultar Singh, Vinod K. Gupta, and Mahesh H. Shah for their help and cooperation.

Thanks are due to Messers. G.L. Misra, S.K. Tewari and S.S. Pethkar for an excellent job in typing. Thanks are also due to Mr. T. Tiwari for an excellent job in cyclostyling.

- Ravindra Prakash

Kanpur 208016

July 1975.

ABSTRACT

A programmable logic controller which uses decision tables as an input language is designed. Programmable logic controllers have wide applicability in controlling machinery and processes. The existing programmable logic controllers use ladder diagrams to describe control. An alternative method of decision tables is proposed to represent the logic. It is felt that this is a convenient method and reduces chances of oversights. A model to represent the decision table control system is developed and should help in evolving decision tables.

The design of programmable logic controller has been documented using a hardware description language. The design can also be adapted to large scale integration using MOS technology.

CHAPTER I

INTRODUCTION

Use of computers in industrial control is well known. The advent of minicomputers has given boost to its use. But still there is a large class of production machines and processes where either these computers are not capable or not cost effective. Nevertheless, these machines are expensive, highly productive and their control is a major concern of manufacturers. This class includes transfer-line machines, dial-index machines, electron-beam welders, gear grinders, stacker cranes, assembly lines, chemical batch process, webprocess (such as glass, films, rubber), petrochemicals and fixed cycles of the numerically controlled machining centres (such as tool or pallet changing).

Until recently such machines and processes were commonly controlled by electro-mechanical relays. Inputs to the relay system came from push buttons, multi-position switches, relay contacts and sensors (such as limit switches). Outputs directed power to motors, solenoids controlling various values & actuators, and signal lights. The internal wiring of the contacts of relays embodied the desired operating sequence. Drawbacks of the relay control circuitry are complex wiring, difficult to incorporate changes and tedious job to locate a fault.

In 1968 General Motors U.S.A. asked controls industry to design an alternate controller which was also programmable. In 1969 Digital

Equipment Corp. responded with a computer like black-box (1), called programmable logic controller abbreviated as PLC. Since then a number of manufacturers have entered into this field.

Programmable logic controllers are small dedicated computers specially tailored to replace relay control circuitry. All the inputs like push-bottons,and sensors are brought into it and terminated there. The voltages at these terminals indicate the status of the respective switches or other elements connected to them. Output lines originate from PLC and provide power to solenoids, motors, signal indicators etc. PLC continuously scans the control program residing in its memory and updates the output lines accordingly. The advantages offered by such controllers are :

Reliability of integrated circuits over that of electro-mechanical relays ;

Ease to Build Control System - The program can be entered into its memory eliminating the complex wiring and can be tested and debugged immediately, saving days and weeks. Program changes can be incorporated easily.

Occupies Less Floor Area - because of compact size which also reduces the cost.

The user can enter the control logic describing how to control the machines or process in ladder diagrams or Boolean equations which are converted into instructions are stored in the memory of PLC by the programming panel.

It was felt that ladder diagrams or Boolean equations are not a good way of describing the control logic because it is difficult to construct a ladder diagram or write a set of Boolean equations and still more difficult to debug them. Therefore we will use an alternate method of 'Decision Tables' to describe control logic.

A decision table shows the rules governing the decision to perform certain actions based on the status of a set of pertinent conditions. This general definition indicates that a decision table can be used to describe any form of decision logic. The intention here is to apply them (set of decision tables) to real time-discrete-control only. The ladder diagram also represents a decision logic and to arrive at it, one must decide about what conditions lead to what actions. Therefore use of decision tables must help one in this thinking process and therefore it was felt that it is unnecessary to construct a ladder diagram if PLC can use decision tables as a input language. Use of decision tables should save momory space as compared to ladder diagrams because decision tables have close relationship with Boolean matrices and may be efficiently stored in computer memory. Ladder diagrams are stored as a sequence of instructions and this program is executed sequentially unless the sequence is altered by a jump instruction. In such a program it is likely that conditions are repeated but in case of a decision table, conditions occur only once. This may also lead to memory saving. It is felt that decision tables are easier to understand than ladder diagrams and so better means of documentation and communication. Moreover decision table format is such that all possible combinations of conditions may be rigorously examined and oversights may be minimized.

The aim of the present work is to first examine some relay systems and analyse the advantages and disadvantages (if any) of using decision tables to represent these. Having done this the next objective was to design a PLC which uses decision tables as the input language and also as the control program to control machinery and processes. The next objective was to investigate if the PLC design lends itself to implementation with existing microprocessor chips. If not whether a new LSI chip or chips could be suggested to do the job.

With this brief first chapter introducing the subject and the problem this thesis has been arranged into the following seven chapters.

Chapter II introduces some representative relay systems. There we have modelled some relay systems and arrived at some basic rules to facilitate the writing of decision tables directly from the word statement of the problem. Finally we have expressed the relay systems as a set of decision tables and evaluated their advantages over the ladder diagram coding of relay systems. In chapter III some constraints are imposed on the decision tables to simplify the design of PLC and to obtain faster execution. Various algorithms to execute decision tables on a computer are examined in chapter IV and an algorithm suitable for hardware implementation is selected. Chapter V contains the description of some data structures and PLC at block diagram level. The complete design of PLC using the hardware description language AHPL (2) is presented in chapter VI. In chapter VII the possibility of designing a PLC with microprocessor chips (LSI) is explored. Thereafter new LSI chips to do the job of our PLC are suggested. The conclusions of the study are presented in chapter VIII.

CHAPTER II

CONTROL LOGIC OF SOME SYSTEMS

Before we attempt to design a programmable logic controller it is necessary that we be clear in our minds about the problems encountered in designing such controllers. Thus it is necessary to look up some representative relay control systems. Thereafter we would express their functions as a set of decision tables and find the good and bad points in the use of decision tables for this purpose.

2.1 PLAN SYMBOLS

The relay engineers use a set of symbols or pictures to represent their systems and communicate with others (3) . There are three major forms of symbols in use. These are shown in the table in Fig. 2.1. The first column is used by the railways, fire alarms, and traffic light plans. The second column is used by the motor control and power control industries and was adopted by the Joint Industry Conference (JIC) as standard electrical symbols. The third column of symbols is attributed to the National Association of Relay Manufacturing (NARM) and is used primarily by the telephone industries. Some of the symbols for relay contacts are shown in the table in Fig. 2.2 again in three columns. The symbols in first column have

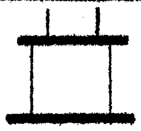
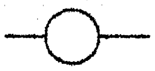
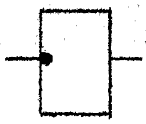
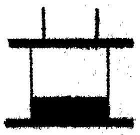
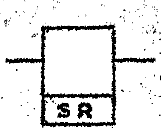
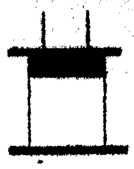
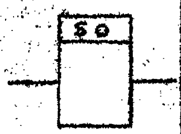
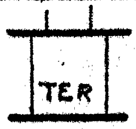
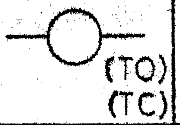
Symbol Name	By Knoop[3]	JIC	NARM
Single Winding Neutral DC Ordinary Acting			
Neutral DC Slow Releasing		—	
Neutral DC Slow Pick Up (Operating)		—	
Time DC			—

FIG. 2.1

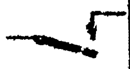
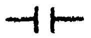
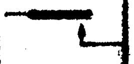
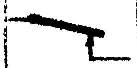
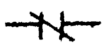
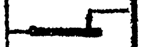
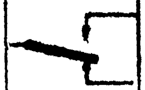
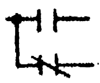
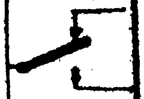
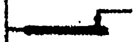
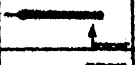
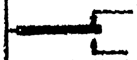
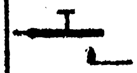
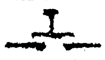
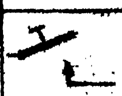
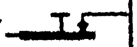
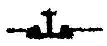
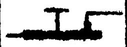
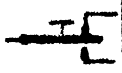
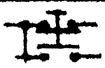
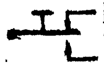
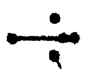
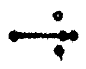
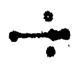
	Symbol Name	[3]	JIC	NARM
Relay Contacts	Front - Deenergized Coil (Normally open) (Make) (Form A)			
	Back - Deenergized Coil (Normally closed) (Break) (Form B)			
	Front - Back - Deenergized Coil (Normally open - Normally closed) (Break - make) (Transfer) (Form C)			
	Front - Energized Coil		—	—
	Back - Energized Coil		—	—
	Front - Back - Energized Coil		—	—
Controlling Devices	Push Button, Spring Return Contact Closed When Pushed			
	Push Button, Spring Return Contact Open When Pushed			
	Push Button, Spring Return Transfer Contact			
	Switch (Lever) Three Position, SPTT, No Spring Return			

FIG. 2.2

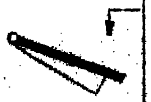
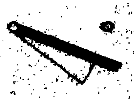
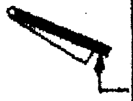
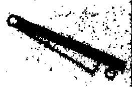
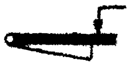
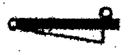
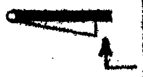
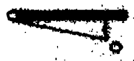
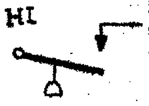
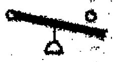
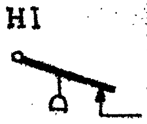
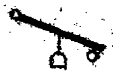
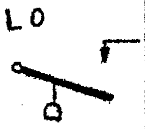
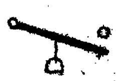
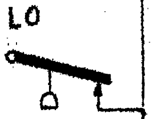
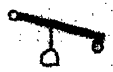
	Symbol Name	[3]	JIC	NARM
limit switches	Front (Normally open) Operating Device Not Against Control Rod			—
	Back(Normally closed) Operating Device Not Against Control Rod			—
	Front(Normally open) Operating Device Holding Control Rod Against Return Spring.			—
	Back(Normally closed) Operating Device Holding Control Rod Against Return Spring			—
pressure switches	Front, Close On High Pressure			—
	Back, Open On High Pressure			—
	Front, Close On Low Pressure			—
	Back, Open On Low Pressure			—

FIG. 2·3

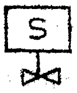
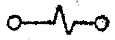
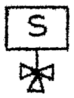
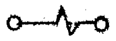
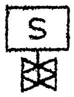
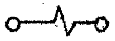
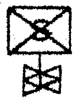
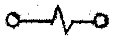
Symbol Name	[3]	JIC	NARM
Two Way Solenoid Valve,DC			-
Three Way Solenoid Valve,DC			-
Four Way Solenoid Valve,DC			-
Four Way Solenoid Valve AC Note:- Two way and Three way AC valves are similar to DC except with "X" in coil			-

FIG. 2·4

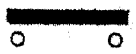
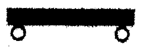
	Symbol Name	Symbol
Indicating contacts	Contact(Open under normal plant condition)	
	Contact(Close under normal plant condition)	

FIG. 2·5

some good properties that they show the condition of contacts at all times, conducting or not conducting. The JIC and NARM symbols show the condition of the contacts only when the relay is not connected to a system or the "on the shelf" condition. In this thesis we will be using the JIC symbols because translation of relay diagram, using them, to decision tables is rather straight forward. The table in Figure 2.3 shows the symbols used to denote indicating contacts, limit switches, and pressure switches. The table in Figure 2.4 shows the symbols for solenoid valves. Figure 2.5 lists two indicating contacts used by us but they do not specifically belong to any association or group of people.

2.2 EXAMPLES OF RELAY SYSTEMS

We have already learned about some symbols used in relay diagrams. Therefore we can study some relay systems. The first example to be studied has been adapted from the book by Knoop (3).

Example 1:

Problem Statement -

A control system is to be designed to control a motor capable of running in two modes viz.

1. Normal
2. Inch.

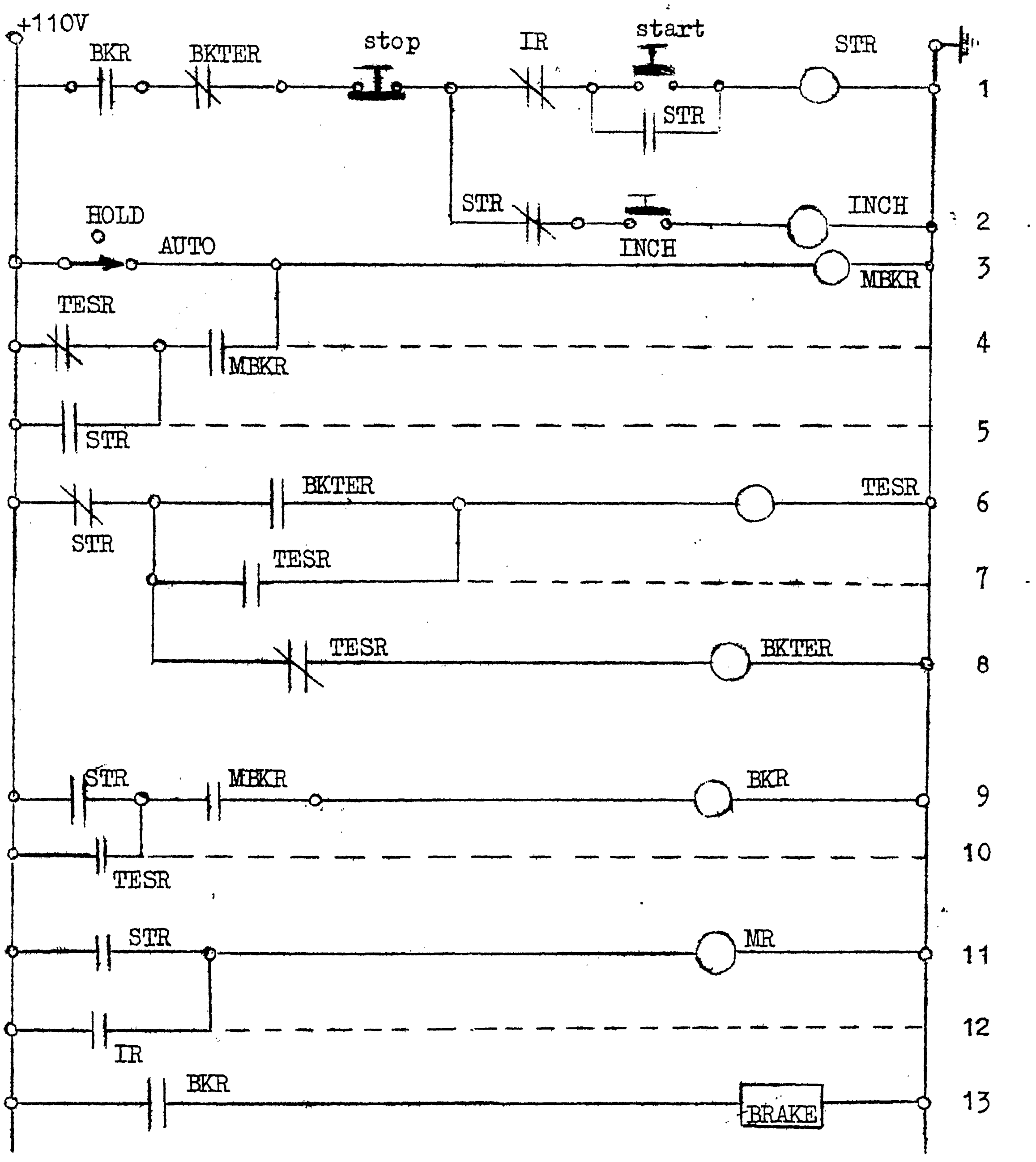
Normal Mode

A start button is provided to start the motor by turning on the power to the motor. If the brakes are in released position (in which case the brakes are in position constraining the motion of the motor) then 'pressing the start push button' should have no effect on the motor. (This is required to protect the motor.) A push button stop switch is provided to stop the motor. As soon as the stop button is pressed, power is turned off from the motor and the brakes are released. After a specified duration of time td the brakes should be lifted up. A DPDT switch HOLD/AUTO is provided to release the brakes manually. While this DPDT switch is in the HOLD position and the motor is not running then the brakes are released. While the motor is running HOLD/AUTO switch should not have any effect on the brakes. Brakes are released if power is turned off to the brakes. If power is applied to the brakes, shoes are lifted up and the motor is free to run.

Inching Mode

Motor should run so long as an INCH button is kept pressed. On the release of the INCH push button motor should coast to a stop without application of brakes. Again if HOLD/AUTO switch is in the HOLD position, motor should not start, on pressing the INCH button.

In both the cases explained above, the switch HOLD/AUTO should have no effect once the motor starts running. This is desired so that unnecessary damage to brakes and motor is not caused by release of brakes to it while it is powered. The relay diagram is given in Figure 2.6.



Relay Diagram for Example 1

Figure 2.6

The readers not familiar with relay circuits may refer to (3). For helping the reader to understand the circuit, the status of relays as a function of the sequence of some of the inputs is tabulated below in Figure 2.7. To start with, it is assumed that the motor is not running and AUTO/HOLD switch is at auto position. IR is throughout in deenergized state and INCH is never pressed.

	Position of Manual switch AUTO/HOLD	State of start P.B.	State of stop P.B.	State of Relays					
				STR	MR	BKTER Timer	TESR	MBKR	BKR
1.	AUTO	R	R	D	D	D	E	E	E
2.	do	P	R	E	E	D	D	E	E
3.	do	R	R	E	E	D	D	E	E
4.	HOLD	R	R	E	E	D	D	E	E
5	AUTO	R	R	E	E	D	D	E	E
6	AUTO	R	P	D	D	T	D	E	D
7	do	R	R	D	D	T	D	E	D
8*	do	R	R	D	D	E	D	E	D
9.	do	R	R	D	D	D	E	E	E

R = For 'released' position of P.B.

P = For 'pressed' position of P.B.

D = For deenergized state

E = For Energized state

T = In state of timing (only for timers@.)

* = State corresponding to 8 is transitory.

@ = It is to be pointed out that there is no direct way to detect this state because contacts of timers remain in a state corresponding to de-energized relay.

Figure 2.7

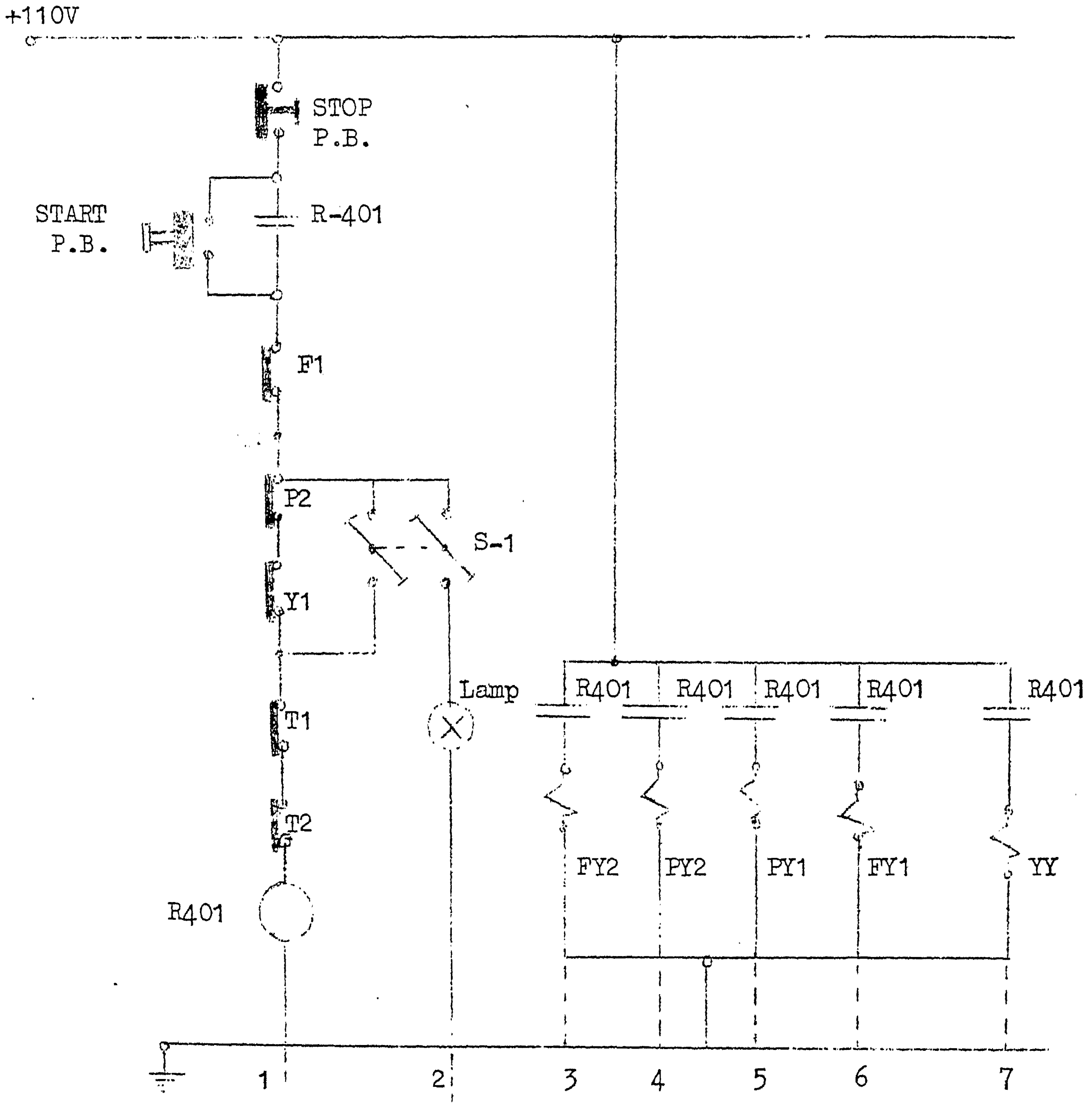

+110V
STOP
P.B.
START
P.B.
R-401
F1
P2
S-1
Y1
T1
T2
R401
Lamp
R401
R401
R401
R401
R401
FY2
PY2
PY1
FY1
YY
1
2
3
4
5
6
7

Figure 2.8

State of the Controlling System - It is the set of states of each relay. A relay can be ordinarily in two states energized or de-energized. In the case of timers, however, they can have one more state namely timing state (T).

From the above example we can see that the next state of the system is not only a function of the input switches and push buttons but also of the state of each individual relay. Therefore the relay system is a sequential system (4).

Example 2 :

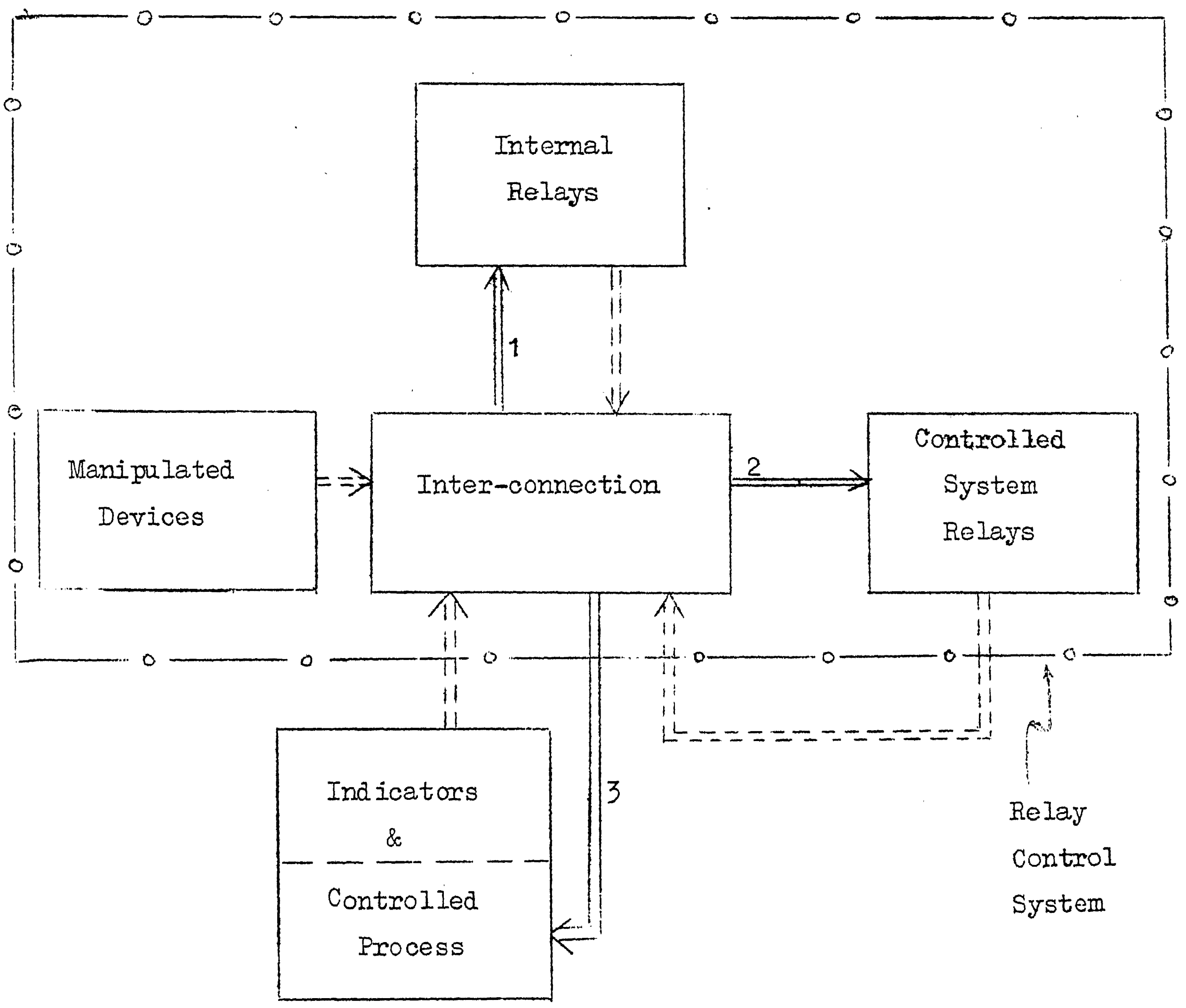
This example has been taken from the relay diagrams supplied by EIL Delhi. Figure 2.8 contains the relay circuit diagram to control a section of Benzene Hydrogenation Plant. The contacts labelled F1, P2, Y1 are indicating contacts for the flow of gas, pressure of a gas, or flame failure respectively. T1 and T2 are indicating contacts for the temperature. The contacts as shown indicate their position when the plant is running under normal conditions. FY2, PY2, PY1, FY1, YY are solenoid valves.

This example brings out another fact that the input to a controlling system or controller constitute another component viz. status indicators of the controlled system. This means there is a feed back from the controlled system to the controlling system and hence the combined system is a closed loop controlled system. We can summarize our findings as follows:

The inputs to a controlling system consists of (1) independent controlling devices (such as push buttons, multiposition switches), (2) status indicators (such as limit switches, pressure switches, temperature switches), (3) contacts of internal relays, and (4) controlling system relays. By the word 'internal relay' we mean those relays which are introduced into the system to remember the states of certain set of contacts and do not control the controlled system directly. The controlling system relays (in addition to forming the inputs to the relay control system) directly control the controlled system.

Relay System Model

We have modelled the relay system to bring out inter-relationship between various components constituting it. See Figure 2.9. The block labelled 'Manipulated Devices' represents independent controlling devices. The block 'Internal Relays' represents internal relays along with all their contacts. The block labelled 'Manipulated Devices' represents independent controlling devices. The block 'Internal Relay' represents internal relays along with all their contacts. The block 'Controlled System Relays' represents the relays whose contacts directly control the process. The contacts also are included in the block. The block 'Indicators and Controlled Process' represents the process (to be controlled) and status indicators of the process. Solid lines denotes the lines going to relays, motors, solenoids, signal indicators etc. If any of these lines is powered then the corresponding relay or any other device connected to it gets the power. If the relay is an ordinary acting one then it gets energized as soon as it gets the power. But if the relay is a timer relay then it gets energized



⇢ From contacts (Information about status of conditions)

⟹ To relays or any other device to be operated or powered.

TOTAL SYSTEM

Figure 2.9

after a fixed interval of time. The dotted lines originate from the two terminals of contacts irrespective of the fact whether they belong to indicating contacts or relay contacts or switch contacts etc. The block 'Interconnection' may be thought of as a patch board which interconnects the incoming lines (dotted) to it and outputs the lines (solid) going to the blocks 'Internal Relay', 'Controlled System Relays and 'Controlled Process and Indicators'. The interconnection embodies the desired behaviour of the relay control system.

It is to be pointed out that there is a relationship between the lines in the ports 2 and 3. The lines in the port 3 can be divided into a number of groups. A group may contain a single line also. Every group except perhaps for one, in the port 3 is directly controlled by a specific line in the port 2. There may be a group in port 3 which is not directly controlled by any single line in the port 2.

To give us faith in our model we have redrawn the relay diagram of Figure 2.8 as Figure 2.10. The break up of various devices into different categories is as follows:

Manipulated Variables	:	START P.B., STOP P.B., S-1
Controlling System Relays	:	R401
Internal Relays	:	Nil
Indicators	:	F1, P2, Y1, T1, T2.

Note R 401 directly controls FY2, PY2, PY1, FY1, YY and Lamp is controlled by S-1.

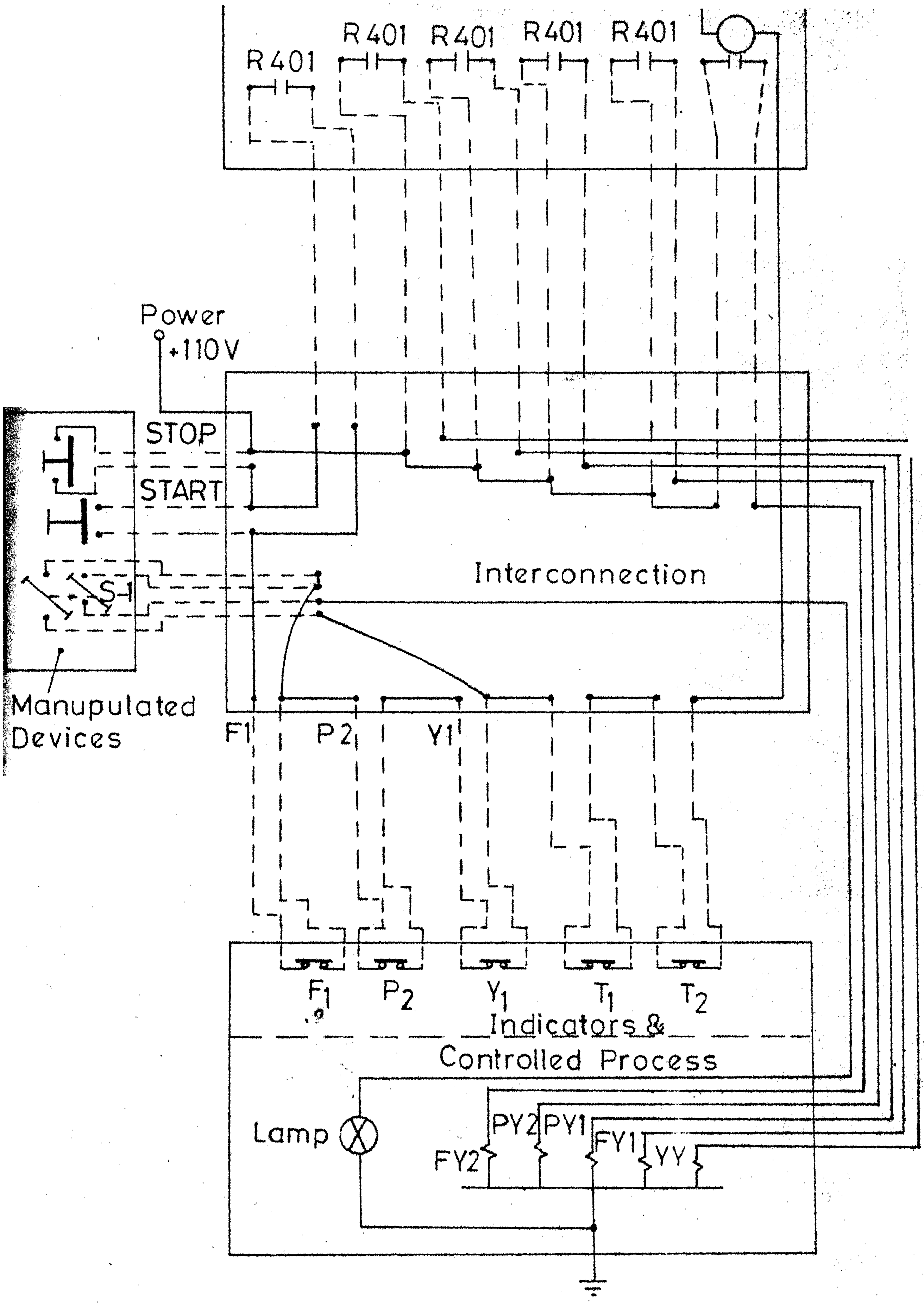
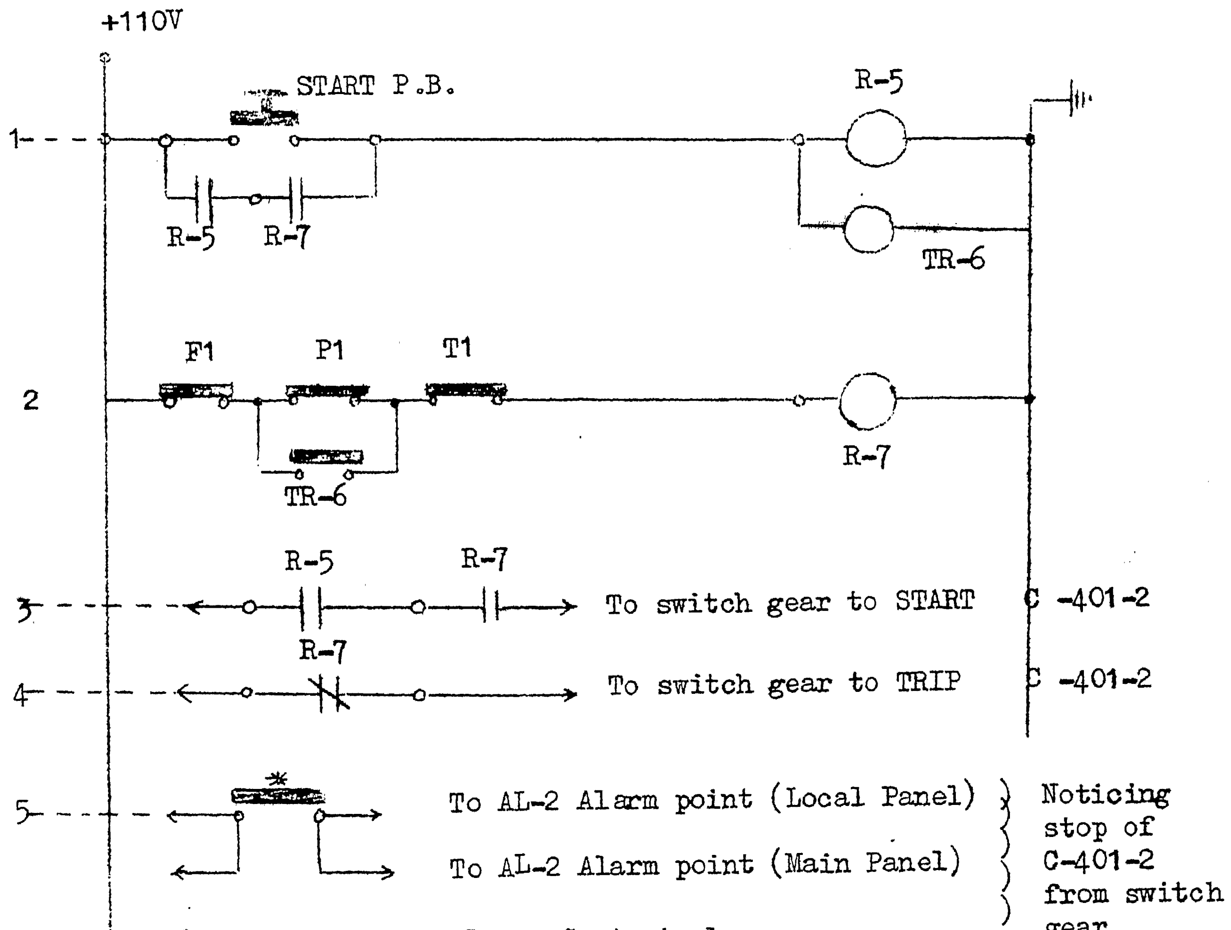

R401
R401
R401
R401
R401
Power
+110V
STOP
START
S-1
Interconnection
Manupulated Devices
F1
P2
Y1
F_1
P_2
Y_1
T_1
T_2
Indicators &
Controlled Process
Lamp
PY2
PY1
FY1
YY
FY2

FIG. 2·10



TR-6 : Time delay Relay - Contact closes on energization and opens after delay adjustable 0 ÷ 60 secs.

* Limit contact from switch gear closes when compressor stops.

Hydrogen Booster Compressor C-401-2

Figure 2.11

We can similarly try the relay system of Figure 2.6. The break up of various devices is as follows :

Manipulated Variables	:	start, stop, INCH, HOLD/AUTO
Controlling System Relays	:	BKR, MR
Internal Relays	:	MBKR, STR, IR, TESR, BKTER
Indicators	:	Nil

Example 3

This example also has been taken from the relay diagrams supplied by EIL Delhi. Figure 2.11 depicts relay control circuitry for Hydrogen Booster Compressor (C-401-2). It is to be noted that the relay TR-6 can be replaced by the basic timer relay of JIC using the following

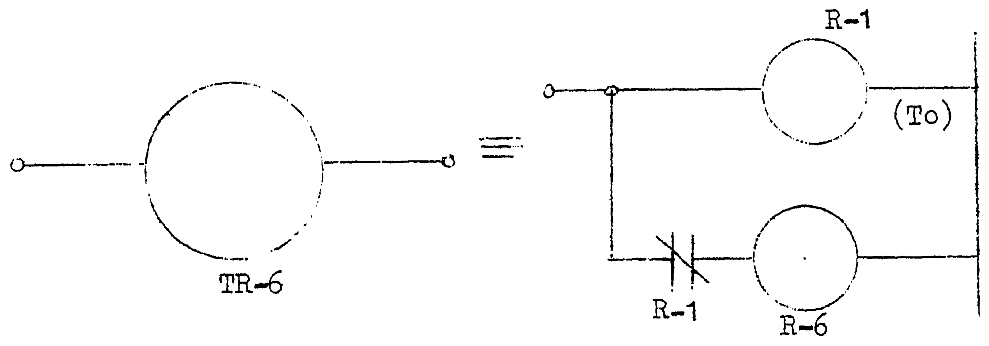


Figure 2.12

where normally open contact of R-6 is to be used to perform the desired function described in case of TR-6.

F1, P1, T1 are limit indicating contacts. Function of timer relay here is to ignore abnormal pressure during start up operation of Booster compressor. Lines 3,4 indicate that start up of compressor depends also upon the status of other contacts from other sections of the plant. Line 5 indicates that the two ends of the contact are fed to an alarm unit on closure of the contact may flash a light and ring a bell.

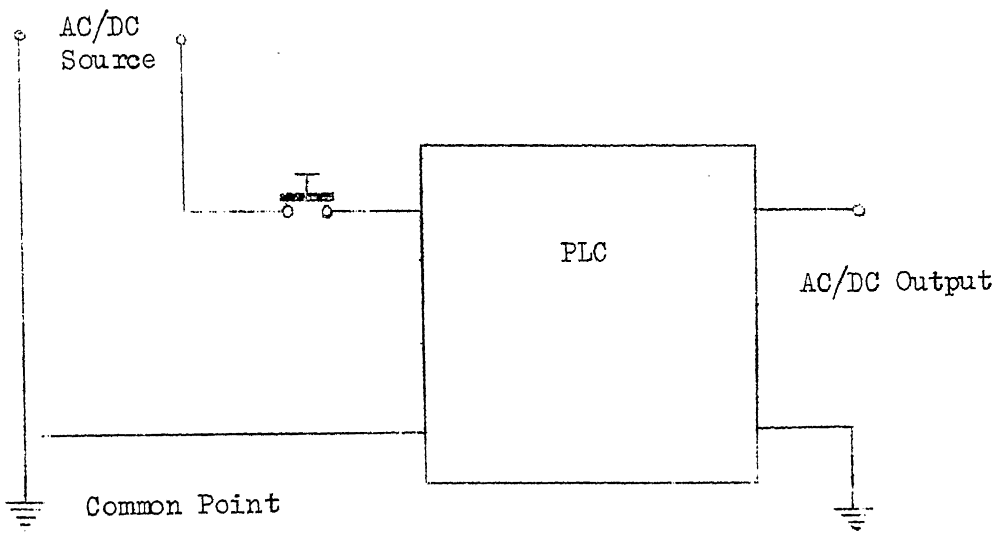


Figure 2.13

2.3 PROGRAMMABLE LOGIC CONTROLLERS

Till now we considered relay systems. Our objective is to replace relay control circuitry by a solid state processor- "programmable logic controller" or PLC in short. Physical inputs to the system will be push buttons, limit switches, pressure switches etc. They can be wired to PLC as shown in Figure 2.13. Hence if any contact closes then PLC sees it as a logical "one" (High voltage level or presence of a voltage is assumed to be logical "one"), otherwise it sees it as a logical "zero".

Output devices, such as solenoids or solenoid valves, motor, any relay etc. can be wired to PLC directly. PLC will have its own interface unit for each line to convert logical 1 and 0 dc. voltage levels to appropriate voltage level of AC or DC type as may be required.

Hence PLC has all binary inputs and binary outputs. And binary outputs are function of inputs, outputs and internal relay contacts. PLC can also be thought of as a decision process which decides the next state of outputs depending upon inputs, previous or present outputs, and internal relay contacts. We hope to represent decision process conveniently by a set of decision tables. Before we go ahead with decision tables we would like to formalize the terminology of decision tables.

Decision Tables :-

We shall confine ourselves to Limited Entry Decision Tables (LEDT) (5). A decision table is divided into 4 parts by double lines as shown in the following Figure 2.14.

Condition Stub	Condition Entries
Action Stub	Action Entries

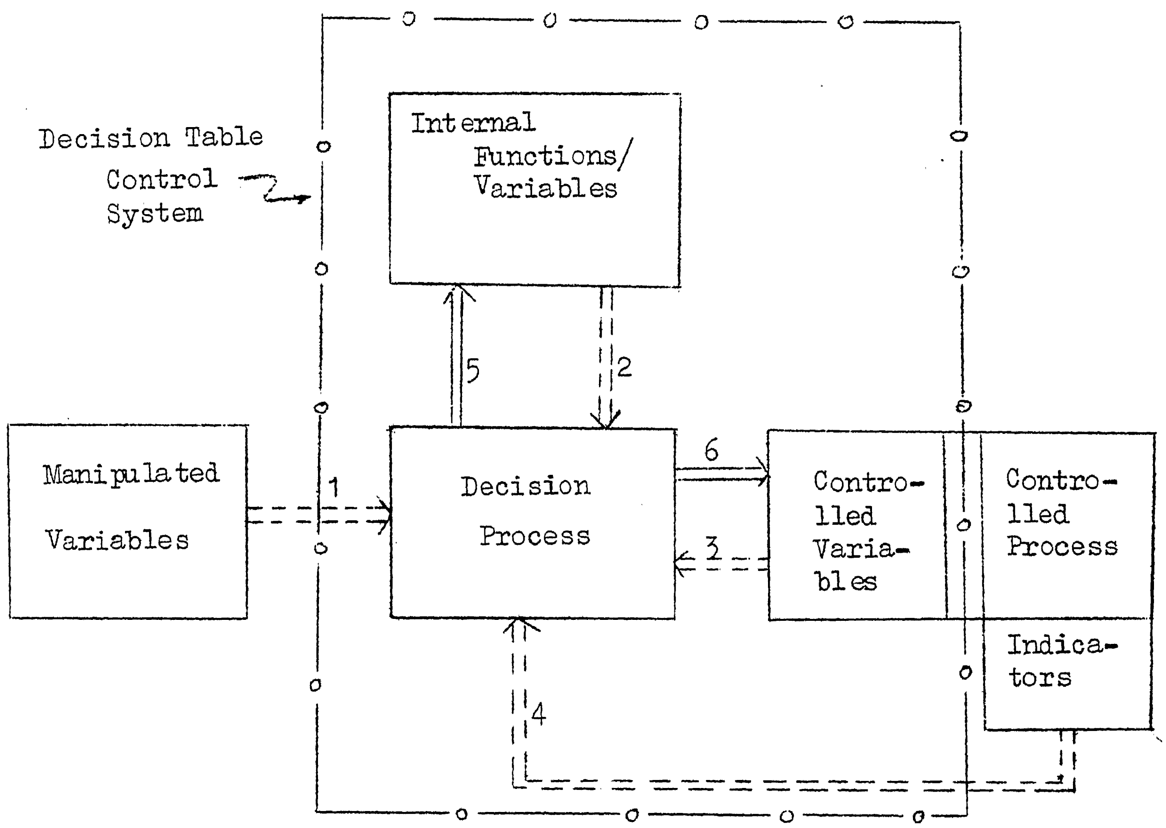
Figure 2.14

The condition stub lists the conditions to be tested and the action stub the actions to be performed. In our examples condition stub would contain conditions like 'a input line = 1', 'a out put line = 0', 'START P.B. = 1' etc. Action stub would contain actions like, 'Energize/Deenergize an output line' or turn on solenoid valve etc. The condition and action entries are divided into a number of columns; each column is known as a rule. (For example see Figure 2.18). In each column, the condition entries consist of Y, or N, or - . Meaning of Y, N, - are that the condition corresponding to the row in which they belong should be true, or false, or irrelavant (also known as don't care). Action entries in a rule are X or -. Action entries X and - tell if a rule is satisfied then the actions corresponding to X are executed and actions corresponding to - entries are not executed. A rule holds if the status of relevant conditions for that rule are same as dictated by the corresponding entries in the rule.

2.4. A SYSTEMATIC APPROACH TO DECISION TABLE WRITING

The following steps are to be followed to arrive at a set of decision tables.

1. Define the problem.
2. Identify all the variables used in problem solving according to the following sequence.
   (i) Manipulated variables (controlling devices).
   (ii) Independent controlling variables.
   (iii) Controlled variables.



- - - - - -> Information going into Decision Process.

═══> Action Paths.

Figure 2.15

(iv) Timers

(v) Counters

(vi) Internal binary variables.

3. Construct decision tables using these variables.

Figure 2.15 which is Figure 2.9 redrawn with modification shows the various variables interacting with the 'Decision Process'. The input port 1 shown in Figure 2.15 constitute the manipulated variables of step 2(i). Examples are START, STOP, INCH, HOLD/AUTO switches. There is no problem in identifying them if problem description is complete.

The variables that come under 2(ii) are port 4 in Figure 2.15. These variables are status indicators of the controlled system and without them automation seems impossible. Their examples are limit switches, pressure switches, presence detectors etc. Port 3 of Figure 2.15 shows the information about controlled variables going into 'Decision Process'. PLC controls the process by manipulating the state of these variable (shown by port 6) in accordance with the controlling program or code residing in its memory. Examples are solenoid valves, indicator lamps etc.

Timer and counters are two components of port 2 of Figure 2.15. They are also known by the name of internal functions. The previous examples 1, and 3 have used timers and their usefulness in control applications is evident. One should not have much difficulty in identifying the number of timers required. The purpose of counters is to count the occurance of certain external events. One should not have much difficulty in figuring them out.

Step 2(vi) requires one to find the internal variables. There is no clear cut method of identifying them. Only a physical understanding of the problem can help one to find them. The procedure would be to guess them, try to write decision tables and if necessary introduce some more internal binary variables. These internal binary variables have some physical significance and are used to remember a type of condition or state of the plant.

We shall now write decision tables for a few problems. Thus we shall have some experience in the art of guessing the internal binary variables.

2.5 CONVERSION OF RELAY SYSTEMS TO DECISION TABLES

Example 1:

Refer to the problem statement of example 1 in sec. 2.2. This constitutes the problem definition. We can divide all the variables into various groups as asked in step 2 of the procedure presented in the last section.

Manipulated Variables	:	START, STOP, INCH, HOLD/AUTO
Independent Controlling Variables	:	Nil
Controlled Variables	:	Motor, Brake Making Motor = 1, means motor has been turned on. Making Motor = 0, means motor has been turned off. Brake = 1 means, brake shoes have been withdrawn. Brake = 0 means, brakes are applied to the motor.
Timers	:	[illegible] Timer
Counters	:	Nil
Internal Binary Variables	:	(to be guessed).

We observe that START is a push button and is used to start the motor under a certain set of conditions. To remember that a START push button was pressed, an internal variable is required. We need not have a separate variable to remember that STOP push button was pressed, because the same variable/state can be flipped. Let this variable be 'STR'. In addition to remembering the old status of START P.B., STR will act as a derived variable which directly tells us whether to run the motor or stop the motor. We need not have an internal variable for INCH P.B. as motor is to be run only if INCH button is kept pressed. Timers may require an internal variable. Need for an internal variable depends upon how we want to use a timer. Before we explain it, let us state the assumptions about the behaviour of timers. It is to be emphasized, in case of relays, for the circuit other than this timer relay, the timer relay has only two states namely energized and deenergized. A timer relay gets energized after a fixed delay after first applying the power to relay. If at any stage power is withdrawn then it gets cleared and then onwards behaves as if no power was ever applied to it. To maintain the same parallelism we also assume that one can test only if 'timer $\geq$ preset value ?' with the answer yes or No. Second assumption is that START·timer is equivalent to powering the relay and hence executing START timer again and again when timer has already been started earlier and has not yet been cleared, will have no effect on the timer. With these assumptions we now analyse when we need an internal variable.

START :

DT NO1	R1	R2	R3	ELSE
C1 : BKR = 1	Y	Y	Y	
C2 : START = 1	Y	-	N	
C3 : STOP = 1	N	N	N	
C4 : INCH = 1	N	-	Y	
C5 : STR = 1	N	Y	N	
Make STR 1	X	X	-	-
Make STR 0	-	-	X	X
Make Motor 1	X	X	X	-
Make Motor 0	-	-	-	X
Make Timrem = 0	X	X	X	-
GO TO DT NO 2	X	X	X	X

DT NO 2	R1	R2	R3	R4	R5	R6	ELSE
C6 : SWITCH = HOLD	-	-	N	Y	N	Y	
C7 : MOTOR = 1	Y	N	N	N	N	N	
C8 :Timer $\geq$ td	-	N	Y	Y	N	N	
C9 : Timrem = 1	-	N	N	N	Y	Y	
Make BKR = 1	X	-	X	-	X	-	-
Make BKR = 0	-	X	-	X	-	X	-
START Timer	-	X	-	-	-	-	-
Clear Timer	-	-	X	X	-	-	-
Make Timrem	-	-	X	X	-	-	-
GO TO DT NO 1	X	X	X	X	X	X	X

Figure 2.16

Let us imagine we are acting in place of PLC. Say at time t_1 STOP is pressed. Then we start the timer, cut off the power to motor and apply brakes to the motor. When timer times out the interval set on it we lift the brakes and clear the timer to zero. Let this occur at time t_2. Now it is evident that status of condition at t_1 and before t_2 is same as after t_2 and will lead to repeating the whole r sequence of applying the brakes and then lifting it. Therefore we need another internal variable say Timrem.

If we decide to operate timer in a different fashion and do not clear at time t_2 then we shall not need an internal variable. In this case timer has to be cleared when START P.B. is pressed. However we decide to operate according to the first method and hence internal variables consists of the following variables:

Internal Binary Variables : STR, Timrem.

Now we shall construct two decision tables for this problem. First one will deal with starting, running, and turning of the motor. The second decision table will take care of brakes (Figure 2.16).

Example 2

Please refer to the example 2 presented in sec. 2.2. No separate problem definition is available. The relay diagram itself is complete specification of the problem. The group up of variables is presented below.

Manipulated Variables	: START P.B., S-1(DPDT), STOP P.B.
Independent Controlling Variables	: F1, P2, T1, T2, Y1

Controlled Variables	:	R 401, Lamp.
Timers	:	Nil
Counters	:	Nil
Internal Binary Variable	:	Nil

We observe from the relay diagram that all the solenoid valves FY2, PY2, PY1, FY1, YY are energized or de-energized simultaneously. Hence we replace all of them by a single variable R 401. 'START' is a push button switch and for reasons similar to those presented in the previous example a internal variable should be required. But when we construct the decision tables we find that both the internal variable and R 401 has always same state and hence the internal variable may be disposed off. R 401 serves also the purpose of remembering the event of pressing the switch. Decision tables are produced in Fig. 2.17.

Example 3

Refer to the Figure 2.11 for the problem specification. The break up of variables into various 6 groups is as follows:

Manipulated Variables	:	START P.B.
Independent Controlling Variables	:	F1, P1, T1, SGCON SGCON stands for switch gear contact.
Controlled Variables	:	STCOM, TRIP-COM We shall assume by making STCOM one, compressor gets started. Similarly by making TRIP-COM one, compressor is tripped off.

Timers	:	Time 1.
Counters	:	Nil
Internal Binary Variables	:	STR

STR is required to remember the event of pressing the START switch. Time variable do not require an internal variable if we let timer remain energized after it times out. (It has been done in original relay diagram. Fig. 2.11). As done in original relay diagram we for the present assume that the alarm unit is available and so switch gear contact will be used to sound the alarm. Therefore there will be no decision table corresponding to alarm unit. Then we have the following decision table (Fig.2.18).

Now suppose that alarm unit is not available but instead a lamp and a bell is available. How to implement function of alarm unit on PLC ? To answer it let us study alarm units. In the market various kind of alarm units with minor variations are available. We shall discuss one.

Depending upon closure (opening) of a contact a flasher bulb starts sending flushes and the bell rings. When an operator notices the alarm he presses 'accept' push button. As a result of it flasher bulb becomes steady and bell silent. If at any stage plant condition becomes normal as indicated by opening (closing) of the contact, bulb turns off and bell goes silent. A switch 'test' function is also provided to test the working of the alarm unit. Turning it on causes a bell to ring and bulb to flash.

Benzene H2/1					ELSE
C1 : START = 1	Y	Y	-	-	
C2 : STOP = 1	N	N	N	N	
C3 : S-1 = 1 (ON)	Y	N	Y	N	
C4 : F1 = 1 (Closed)	Y	Y	Y	Y	
C5 : P2 = 1 (")	-	Y	-	Y	
C6 : Y1 = 1 (")	-	Y	-	Y	
C7 : T1 = 1	Y	Y	Y	Y	
C8 : T2 = 1	Y	Y	Y	Y	
C9 : R401 = 1	N	N	Y	Y	
Make R 401 = 1	X	X	X	X	-
Make R 401 = 0	-	-	-	-	X
Go to Benzene H2/2	X	X	X	X	X

Benzene H2/2			ELSE
C1 : START = 1	Y	-	
C2 STOP = 1	N	N	
C3 S-1 = 1	Y	Y	
C4 F1 = 1	Y	Y	
C9 : R 401 = 1	N	Y	
Make Lamp = 1	X	X	-
Make Lamp = 0	-	-	X
Go To Benzene H2/1	X	X	X

Figure 2.17

DT NO 1	R1	R2	R3	R4	ELSE
C1 : START = 1	Y	-	-	Y	
C2 · STR = 1	N	Y	Y	N	
C3 · F1 = 1	Y	Y	Y	Y	
C4 · P1 = 1	-	-	Y	Y	
C5 · T1 = 1	Y	Y	Y	Y	
C6 : Time1 $\geq$ td	N	N	Y	Y	
Make STR = 1	X	X	X	X	-
" STCOM = 1	X	X	X	X	-
" STR = 0	-	-	-	-	X
" STCOM = 0	-	-	-	-	X
START Time 1	X	X	X	X	-
Clear Time 1	-	-	-	-	X
Make TRIPCOM = 1	-	-	-	-	X
" TRIPCOM = 0	X	X	X	X	-
Go To DT No 1	X	X	X	X	X

Figure 2.18

Implementation of example 3 with alarm unit -

To generate flashing operation, bulb has to be turned on for a duration say td_2 and then turned off for a duration say the same one td_2. The above sequence of turning 'on' and 'off' has to be repeated. One way to solve this problem is to use two timers. Other way is to have one timer and one internal variable. The second method will be demonstrated. All the variables are divided into 6 groups as follows :

Manipulated Variables	: START P.B., accept P.B., switch Test.
Independent Controlling Variables	: F1, P1, T1, SGCON
Controlled Variables	: STCOM, TRIPCOM, Lamp, bell.
Timers	: Time 1, Time 2.
Counters	: Nil
Internal Binary Variables	: acrem, var 1, STR

The reason for introduction of acrem will be obvious. (To remember accept push button). var 1 is the internal variable introduced along with Time 2.

Decision table 1 remains same except 'Go To' action is no more there. Two more decision tables are added up. See Figure 2.19.

We have introduced a new type of action 'PERFORM' a table. This allows one to easily break up the decision tables into smaller ones. Some times repetition of conditions can be eliminated by use of 'PERFORM' action.

2.6. CONCLUSION

One can go about designing a PLC which stores these decision tables in a suitable form and then executes them. Before we do so, we must analyse the implications of doing this. Is it worth-while

ALARM : DT NO 2					ELSE
C1 : SGCON = 1	-	Y	Y	Y	
C2 : accept = 1	-	N	Y	-	
C3 : acrem = 1	-	N	N	Y	
C4 : Test = 1	Y	N	N	N	
PERFORM FLASHER	X	X	-	-	-
Make bell = 1	X	X	-	-	-
Make bell = 0	-	-	X	X	X
Make Lamp = 1	-	-	X	X	-
Clear Time 2	-	-	X	X	X
Make acrem = 1	-	-	X	X	-
Make Lamp = 0	-	-	-	-	X
Make acrem = 0	-	-	-	-	X
Go To DT NO 3	X	X	X	X	X

FLASHER : DT NO 3				
Time 2 $\geq$ td_2	N	Y	N	Y
var 1 = 1	N	N	Y	Y
START Time 2	X	-	X	-
Clear Time 2	-	X	-	X
Make var 1 = 1	-	X	X	-
Make var 1 = 0	-	-	-	X
Make Lamp = 1	X	-	-	X
Make Lamp = 0	-	X	X	-
Go To DT No 1	X	X	X	X

Figure 2.19

to express the word statement of a problem in decision table format and implement the tables using PLC's or is it better to obtain a relay ladder diagram and use it in PLC's ? The author feels that the first approach i.e. to obtain decision tables from word statement of the problem has definite advantages over the other one.

(1) To arrive at a ladder diagram, for each relay or each out-put line, one has to decide about all the contacts affecting the relay or the out-put line and about their (contacts) interconnections. Let us consider a single relay or a single out-put line. The interconnections generally turns out to be a tree structure. It is not possible except for trivial situations to just drawout tree network of contacts in one stroke because one has to concentrate on positioning of contacts, their status (normally open or normally closed) and wiring. One has to go through many trials to arrive at a reasonable ladder diagram and then justify that there is no condition of contacts leading to undesirable state of the relay. Whereas in case of decision tables, all the conditions in a decision table can be listed and then the rules leading to the desired actions may be filled out one by one. Decision table structure lets one concentrate on one rule at a time. It further lets him denote his full thought in only the decision process and does not distract his attention in other activities like positioning or wiring. It is known that a decision table can be converted to relay tree network (6) . Therefore attempting to draw a ladder diagram is equivalent

to (perhaps more than that) envisaging all the condition entries of the corresponding decision table and hence a difficult task as compared writing a decision table.

(2) The decision table format lets one examine all the possible combinations of plant conditions and so oversights are minimized.

(3) The decision table enforces clear thinking in the mind of the writer and so . incomplete problem statement may be pointed out.

(4) The decision tables are superior form of documentation for communication among personnel.

Therefore it is a worth while effort to design a PLC which uses decision tables (as applied to relay systems) as the input language.

CHAPTER III

DECISION TABLE FORMAT FOR PLC

In this chapter the author is going to put some constraints on the decision table structure. A definite meaning will be assigned to this constrained decision table structure. Specification of decision table follows.

3.1 RULES FOR WRITING THE DECISION TABLE

1. Maximum number of conditions = 7
2. No limit on the number of rules
3. Only following type of conditions are allowed
   (i) An input line = 1
   (ii) An output line = 1
   (iii) An internal binary variable = 1
   (iv) Counter (an internal function) $\geq$ preset count
   (v) Timer (an internal function) $\geq$ preset value
   In case of timers preset value will be discretized to 0.1 secs.
4. There will be no action enteries. Decision table will look like:

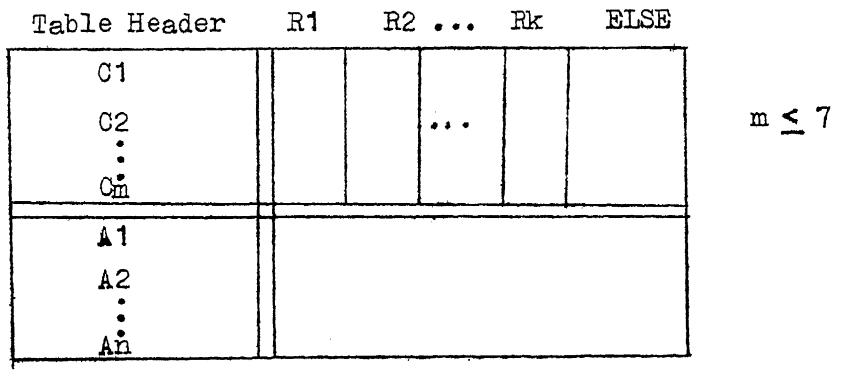


Figure 3.1

5. Meaning assigned to the table is as follows. (complement of action is defined later):

(i) If any one or more of the first R1 to Rk rules are satisfied then the actions to be executed are A1,A2,...An provided no action 'causes' PLC to jump to a new decision table. If an action causes it to jump to another decision table remaining actions in the action stub are not executed.

(ii) In case none of the R1 to Rk rules are satisfied (i.e. ELSE rule holds) the actions to be executed are complements of A1,A2,... An provided no complemented action causes it to jump to another decision table. Complement of 'GoTo' action is assumed to be "No Operation".

6. List of Actions Allowed and their Meanings

(i) A1: Make an internal binary variable or output variable '1'.

(ii) A2: Make an internal binary variable or output variable '0'.

Complement of A1 is defined to be A2 and vice-versa.

(iii) A3: Start a timer.

(iv) A4: Clear a timer.

Complement of A3 is defined to be A4 and vice-versa. Timer once started will time till it reaches a preset value. It stops at a preset value and remains there. However a timer can be cleared to zero at any stage by executing A4 action. If a timer is timing or has reached a preset value, action A3 has no effect on it.

(v) A5: INC (increment) a counter.

(vi) A6: DEC (decrement) a counter.

When A5 or A6 is executed counter is incremented or decremented provided certain condition/conditions is/are satisfied. The condition common to A5 and A6 is (i) 'counter is in enabled state'. The condition unique to INC or A5 is that (ii) counter has not reached a preset count. What is condition (i) and why does it arise? Before we answer it we have to look into the way PLC functions.

It is clear that PLC controls the process according to decision tables stored in its memory. It begins from the first decision table, examines the conditions in it, finds the rule satisfied and then acts on the actions. It then takes the next decision table in sequence and repeats the above steps of examining, finding and acting. This way it races through all the decision tables and restarts the procedure from the first decision table. PLC acts fast enough so that it can race through all the decision tables many times in a second to give the effect of instantaneous response of a corresponding hardwired relay controller. The racing through all decision tables once is called a single 'scan'. It is thus quite likely that the external event which caused the counter to be incremented or decremented, persists for 2nd, 3rd and subsequent scans of the decision table program. Hence some means must be sought to prevent <u>multiple counting of an event</u>.

The solution is to provide two states to the counter namely (1) Enabled state and (2) Disabled state . Suppose a counter is in 'enabled state'. As soon as an event causes action A5 or A6 to be executed, alter the content of the counter and transfer it to the disabled state. Let the counter remain in disabled state in subsequent scans unless an ELSE rule in the same decision table is satisfied. The holding of ELSE rule implies that the event is no more there. Therefore we should now change the state of the counter from disabled state to enabled state. In the subsequent scans if theELSE rule is satisfied again let the counter remain in enabled state unless a non ELSE rule is satisfied. Therefore the following meaning has been assigned to actions A5 or A6.

Action A5:

(By definition of decision table action A5 is executed only if rules R1,R2...Rk are satisfied). Counter is incremented only if counter is in enabled state and counter value is less than preset count. State of counter is made disabled if it was in enabled state.

Action A6:

Counter is decremented only if counter is in enabled state. State of the counter is made disabled if it was in the enabled state.

<u>Complement of A5 or A6:</u>

(Note: Complement of A5 or A6 is executed only if ELSE rule is satisfied) If the state of counter was disabled then its state is changed to enabled state, otherwise no alteration of any kind is made.

(vii) A7: CLR (clear) Counter :

It means clear the counter to zero count irrespective of its state.

Complement of CLR - is "no operation".

(viii)A8: GDE (GO to Decision Table if ELSE rule)

If GDE is to be executed that is, atleast one of the rules R1 or R2 or...Rk holds then the action is treated as no operation. If complement of GDE is to be executed that isELSE rule holds then control is transferred to the decision table mentioned. The actions, if any are left in the action stub, are not executed.

(ix) A9: GDT (GO to Decision Table)

If GDT is to be executed i.e. atleast one of the rules R1 through Rk holds, then the decision table mentioned is taken up for execution. If any actions are left in the action stub, they are not performed.

The complement of GDT is "no-operation" instruction.

7. There is no limit on the number of actions.

3.2 SOME ANALYSIS OF OUR DECISION TABLE

Observation (1) - Each decision table can be thought of as representing a Boolean equation.

A rule Ri is satisfied if $C_{\alpha i} \cdot C_{\beta i} \cdots C_{\delta i} = 1$ where $C_{\alpha i}$, $C_{\beta i}$ etc. represent complemented or uncomplemented relevant conditions for that rule. If in a row and column Ri , Y entry is there, that row variable (condition) appears uncomplemented. If N entry is there, that row variable (condition) appears complemented. If - entry is there, that row variable (condition) is not relevant. Now if at least one of the rule R1 through Rk is satisfied then actions to be executed are (A1,A2...An). The condition for execution of (A1,A2,...An) is

$$\ell = C_{\alpha 1} \cdot C_{\beta 1} \cdot \cdots C_{\delta 1} + C_{\alpha 2} \cdot C_{\beta 2} \cdot \cdots C_{\delta 2} + \\ + C_{\alpha k} \cdot C_{\beta k} \cdot \cdots C_{\delta k} \qquad (1)$$

If $\ell = 1$ the actions (A1,A2...An) are executed.

If $\ell = 0$ then the actions ($\overline{A1}$, $\overline{A2}$ ... $\overline{An}$) are executed.

It should be noticed that action vectors (A1,A2,...An) or ($\overline{A1}$,$\overline{A2}$,...$\overline{An}$) may alter some of the conditions in the right hand side of the equation (1) This makes the total system a <u>sequential</u> logic system.

<u>Observation</u> (2) - Each relay in a relay system represents a Boolean equation.

Consider a relay in the relay system. We know that a relay is connected through a tree of contacts to power supply and ground. Hence a Boolean variable which tells if relay gets energized or not is

$$e = t_{\alpha 1} \cdot t_{\beta 1} \cdots t_{\delta 1} + t_{\alpha 2} \cdot t_{\beta 2} \cdots t_{\delta 2} + \\ + t_{\alpha p} \cdot t_{\beta p} \cdots t_{\delta p} \qquad (2)$$

where $t_{\alpha j}$ is some contact in the tree. If contact is closed then $t_{\alpha j} = 1$.

If contact is open then $t_{\alpha j} = 0$.

The relay which is governed by variable e in the above equation, may have some of contacts in the right hand side of the equation. This makes the equation representing a sequential logic. More general situations like relay R1 affecting relay R2, relay R2 affecting R3 and so on in the end affecting Rp which in turn affects R1, also make the total relay system a sequential one.

Inference : Similarly between equations (1) and (2) is obvious. Differences are the following:

(1) In equation (1) a action vector is executed whereas in case of equation (2) a single action is performed.

(2) The action set of equation (1) are more powerful.

(3) In case of equation (1) there is a limitation on the total number of different variables (ignoring the difference of complementation) on the right hand side.

It is to be stated that equation (2) can always be broken down into number of equations satisfying the constraints mentioned in (3) above.

3.3 AN EXAMPLE

An example will be worked out using the new decision tables. The problem taken is the same as the one presented in chapter II in Example 3. The reader might then compare both sets of decision tables. For the reader's convenience, all the variables with groupings are produced below.

Manipulated Variable	: START P.B., accept P.B., switch Test;
Independent Controlling Variables	: F1, P1, T1, SGCON;
Controlled Variables	: STCOM, TRIP-COM, Lamp, bell;
Timers	: Time 1, Time 2;
Counters	: nil;
Internal Binary Variables	: acrem, var1, STR;

When START is pressed we start the timer, make STCOM, and STR both one, make TRIP-COM zero and then wait for timer to time out the interval. Hence* actions corresponding to STCOM, TRIP-COM, STR, Time 1 can be grouped together into one decision table

DTNO 1					
C1: START = 1	Y	-	-	Y	ELSE
C2: STR = 1	N	Y	Y	N	
C3: F1 = 1	Y	Y	Y	Y	
C4: P1 = 1	-	-	Y	Y	
C5: T1 = 1	Y	Y	Y	Y	
C6: Time1 $\geq$td1	N	N	Y	Y	
Make STR = 1					
Make STCOM= 1					
Make TRIP-COM=0					
START Time1					

Fig. 3.2

* As we have given a fixed meaning to "action stub" and "holding of a rule" only those actions which respond in 'same' manner to a set of conditions can be grouped in a single decision table.

Note that execution of ELSE rule causes STCOM, STR to become zero, time1 to be cleared, and TRIP-COM to become one. An alarm unit is assumed to be available in the above solution. Alarm unit responds to the contact SGCON.

Suppose now alarm unit is not available but a lamp and a bell are available. (For the description of alarm unit see chapter II example 3). The conditions to which bell and lamp respond are different. So we will have separate decision tables to handle ring and lamp. The common conditions are SGCON, accept, acrem, test. It is usually preferable to process common conditions in separate decision tables and store their statuses in a few intermediate variables. But here we observe that 'bell' responds only to above conditions and status of bell itself will give as sufficient infomation about their statuses. Therefore we dispense with the intermediate variable. Decision table 1 remains unaltered. Decision table 2 is constructed in Figure 3.3.

DT No 2			
C1 : SGCON = 1	-	Y	ELSE
C2 : accept = 1	-	N	
C3 : acrem = 1	-	N	
C4 : Test = 1	Y	N	
bell = 1			

Figure 3.3

bell = 0 means lamp should be either off or steady.

bell = 1 means lamp should flash.

As in chapter 2 we decide to generate flash by using a single timer. Our scheme to generate flash is to start timer. (Assume all variables are cleared to zero before start of execution of set of decision tables. Therefore lamp is initially off.) When it times out, clear it, turn on the lamp, and start timer once again. Then wait for timer to time out. When it times out, clear it, turn off the lamp and start timer again. This process is repeated to give the flashing lamp. One can see that separate tables are required for timer and lamp. One solution, which one might be tempted to give, ispresented Figire 3.4

DT NO 3		
bell = 1	Y	ELSE
Time2 $\geq$ td2	N	
Start Time2		

DT NO 4				
C1 : bell = 1	Y	Y	N	ELSE
C2 : SGCON = 1	-	-	Y	
C3 : Lamp = 1	Y	N	-	
C4 : Time $\geq$ td2	N	Y		
Lamp = 1				

Figure 3.4

But there is a flaw in above decision tables. Assume initially lamp to be on. Let Time2 < td2 and bell = 1. Rule R1 of DTNO4 holds. Now suppose timer times out before execution of DT NO 3. ELSE rule of DT NO 3 holds and so timer is cleared and again when

DT NO 4 is executed contrary to our expectation rule R1 holds. Therefore the Lamp is not turned off. This occurs because DT NO 4 fails to notice the change in the state of the timer. State of the timer is restored before DT NO 4 notices it. (Such situations should be carefully avoided). The solution to the above problem is to introduce an intermediate variable. Let this variable be var1. The correct solution is given in Figure 3.5.

DT NO 3		
C1 ; bell = 1	Y	ELSE
C2 : Time2 $\geq$ td2	N	
Start Time2		
Make var1=0		

DT NO 4				
bell = 1	Y	Y	N	ELSE
SGCON = 1	-	-	Y	
Lamp = 1	Y	N	-	
var1 = 1	N	Y	-	
Make Lamp=1				

Figure 3.5

Comment

Decision tables by definition represent a multi-input and multi-output logic systems. We have restricted its structure (action entries are not there and ELSE rule has a definite meaning) as we do not need the full power of multi-input, multi-output decision tables to represent relay control systems.

Relay control systems are such that one set of conditions which affect one variable and another set of conditions which affect another variable, do not have sufficient number of common conditions to justify their inclusion into one multi-input/multi-output decision table. If we did put them together in one decision table, total storage requirement for this decision table exceeds the storage requirement of the two previous decision tables. A case in point is the example solved in this chapter. Moreover considerable time is wasted in searching through action entries; (multi-input/multi-output decision tables have action entries). The algorithm to execute tables becomes more complex and requires pointers to be maintained, leading to further increase in the overall execution time of tables. Therefore the restrictions imposed on decision tables have made them more useful.

CHAPTER IV

CONVERSION ALGORITHMS

Algorithms for the conversion of decision tables to computer programs fall into two main classes : tree methods (7,8) and mask methods (9,10,11). The criteria for selecting a algorithm in our case would be

1. storage space of objective program,
2. execution time, and
3. simplicity of the algorithm.

Our aim is to design a PLC which executes this object program. PLC should be simple so that its cost is considerably lower than that of a minicomputer.

Tree methods have the disadvantage that they require complex data structure. Lots of storage is required to store branches (links). Moreover space is wasted in repeating the conditional information many times (Figure 4.1). Additional computer facility is required to convert decision tables into the object program . This will also mean that "on line" changes in the decision tables can not be incorporated. Therefore we outright reject the algorithms (7,8) as not suitable for us.

DT EX.	R1	R2	R3
C1	N	N	Y
C2	N	-	Y
C3	N	Y	N
A1			

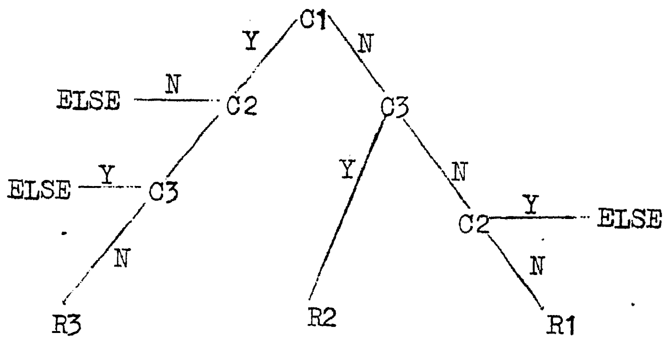


Figure 4.1

Algorithm due to Kirk

The algorithm (3) was presented by H.W. Kirk. The technique described in this paper is based on the creation of a binary image of a limited entry decision table in computer memory. A binary image of a given set of input conditions can also be created. This data image is used to scan the decision table image to arrive at the proper course of action. The algorithm can be best illustrated by an example.

Consider the decision table in Figure 4.2

Step 1. Two matrices named mask matrix M and decision matrix D are to be constructed from the table. The mask matrix M is obtained by representing conditions (the Y's and N's) by 1's and non-pertinent or don't cares (-) conditions by 0's. The decision matrix D is obtained by representing the yes conditions by 1 and all others by 0. These matrices are shown in Figure 4.3. These matrices are called binary images of the decision table.

	R1	R2	R3
C1	Y	Y	N
C2	Y	-	-
C3	N	Y	-

Figure 4.2

$$\underset{M}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}} \qquad \underset{D}{\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}$$

Figure 4.3

Step 2. A data vector is constructed by evaluating the conditions. Conditions are evaluated one by one. If we are evaluating the ith condition and it is true then 1 is placed at ith position in the data vector. Otherwise 0 is placed at the corresponding position.

$$\underset{\text{data vector}}{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}} \qquad \begin{array}{l} C1 = \text{TRUE} \\ C2 = \text{FALSE} \\ C3 = \text{FALSE} \end{array}$$

Figure 4.4

Step 3. (Scanning phase)

Logically 'AND' the particular column in the M matrix with the data vector and then compare the result with corresponding column in D matrix for equality. If equality holds that rule (column) is satisfied otherwise the rule do not hold. Try another rule. Figure 4.5 demonstrates the operation. If no column satisfies the condition of equality then ELSE rule holds.

Rule	Data Vector		Masking Column	=	Result	Corresponding D column	Equal
1	$\begin{matrix}1\\0\\0\end{matrix}$	AND	$\begin{matrix}1\\1\\1\end{matrix}$	=	$\begin{matrix}1\\0\\0\end{matrix}$	$\begin{matrix}1\\1\\0\end{matrix}$	NO
2	$\begin{matrix}1\\0\\0\end{matrix}$	AND	$\begin{matrix}1\\0\\1\end{matrix}$	=	$\begin{matrix}1\\0\\0\end{matrix}$	$\begin{matrix}1\\0\\1\end{matrix}$	NO
3	$\begin{matrix}1\\0\\0\end{matrix}$	AND	$\begin{matrix}1\\0\\0\end{matrix}$	=	$\begin{matrix}1\\0\\0\end{matrix}$	$\begin{matrix}0\\0\\0\end{matrix}$	NO

Application of Step 4 : ELSE rule satisfied.

Figure 4.5

The Algorithm due to Muthukrishnan and Rajaraman (11)

This mask algorithm handles decision table in a elegant way. This algorithm will be explained with reference to the decision table in Figure 4.6.

C1	Y	-	N
C2	-	N	Y
C3	N	N	-

Figure 4.6

Step 1. Generate a matrix M by substituting the following codes for the condition entries in the given table :

$$Y \rightarrow \begin{matrix}1\\0\end{matrix} \quad , \quad N \rightarrow \begin{matrix}0\\1\end{matrix} \quad , \quad - \rightarrow \begin{matrix}1\\1\end{matrix}$$

By substituting the above codes in the decision table of Figure 4.6, we obtain

$$M = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Step 2. For a given record, the conditions are tested and results are stored in a column vector d by using the same codes as in step 1. For instance if tests on some data reveal that C1 is FALSE, C2 is TRUE and C3 is TRUE,

$$d = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} .$$

Step 3. Select the rows of M which correspond to the 1 entries in d and logically 'AND' the resulting elements in a given positions. For clarity, d may be adjoined to the matrix M. Let the resulting row vector be denoted R.

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \begin{array}{rccc} & 0 & 1 & 1 \\ & 1 & 0 & 1 \\ & 0 & 0 & 1 \\ \hline R = & 0 & 0 & 1 \\ \text{element position} = & 1 & 2 & 3 \end{array}$$

Step 4. If all the elements of R are 0, the ELSE rule applies. If the ith element in R is 1 and all other elements are 0, the ith rule applies. If more than one element of R is 1, all the rules corresponding to 1 elements in R being simultaneously satisfied.

In above example R is [0 0 1] , rule 3 holds.

Comparison of above two algorithms

1. The two algorithm are different in the way they code Y, N, and - entries. By interleaving the rows of two matrices M and D of the algorithm (9), we see that Kirk's scheme is essentially

$$Y \rightarrow \begin{matrix}1\\1\end{matrix}, \quad N \rightarrow \begin{matrix}1\\0\end{matrix}, \quad - \rightarrow \begin{matrix}0\\0\end{matrix}$$

2. The storage requirements for both algorithms are essentially same. The data vector for first is half the size of the data vector d of the second . It is to be pointed out that data vector can be shared by all the decision tables in the system. In view of this, this difference is immaterial.

3. The algorithm due to Kirk takes two steps to find out whether a rule is satisfied or not. In case of algorithm due to Muthukrishnan and Rajaraman it is not so. (The later algorithm (11) can be modified to operate it rulewise).

4. Two matrices are required to be stored per decision table in Kirk's method and hence pointers will be required to keep track of particular columns. This will increase the complexity of the conversion program.

5. Suppose decision tables are stored as matrices in computer memory. Suppose we want to make some changes in a decision table. In the case of the algorithm due to Kirk, it is more difficult because there are two matrices per decision table. This is important in the context that PLC is not going to be a minicomputer and debugging facilities are to be provided.

Hence due to the reasons under paras 3,4,5 above, we prefer to use algorithm (11) over algorithm (9). We would now mention some features about the algorithm (10).

Algorithm (10)

This algorithm due to King users "Interrupted Rule Mask" procedure. Therefore it requires a preparation of interrupted rule mask and consequently more storage space. The program is also more complex which would tend to increase the execution time. Saving in time is achieved by not testing irrelevant conditions. Increase in the complexity of the algorithm does not justify its use.

The author has found out (by studying a number of examples) that the number of conditions usually exceeds the number of rules in a decision table constructed for process control applications. The same findings have been reported by G.W. Oerter (12). In view of this it is better if we process decision tables rule wise and this has the following advantage :

(1) As soon as a rule is satisfied we can quit to the action execution phase and hence in many cases we will not have to examine each rule or all the condition entry codes. This will result in faster execution.

We shall therefore modify the algorithm (11) to suit our purpose.

Modified Algorithm :-

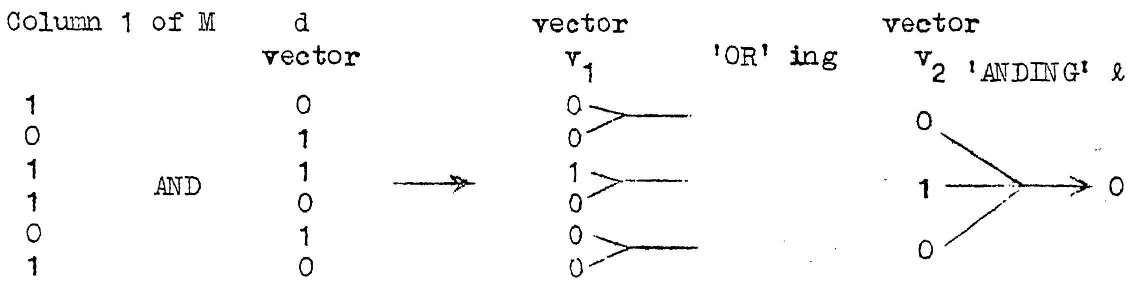
Step 1. Same as step 1 of the algorithm (11).

Step 2. Same as step 2 of the algorithm (11).

Step 3. Fetch a particular column of M matrix. 'AND' the d vector with this fetched column. Let the resultant vector be called v_1. Get a reduced vector v_2 of half the size of v_1 by 'OR' ing the elements of vector v_1 at positions 1 with 2, 3 with 4, 5 with 6,and so on. 'AND' all the elements of this reduced vector v_2 to get a single element 'ℓ'. If this element $\ell = 1$ then the rule (column) is satisfied, otherwise this rule does not hold.

If this rule is not satisfied, repeat above procedure with another column (which has not been tried out). If all the columns of matrix M have been tried out then ELSE rule applies.

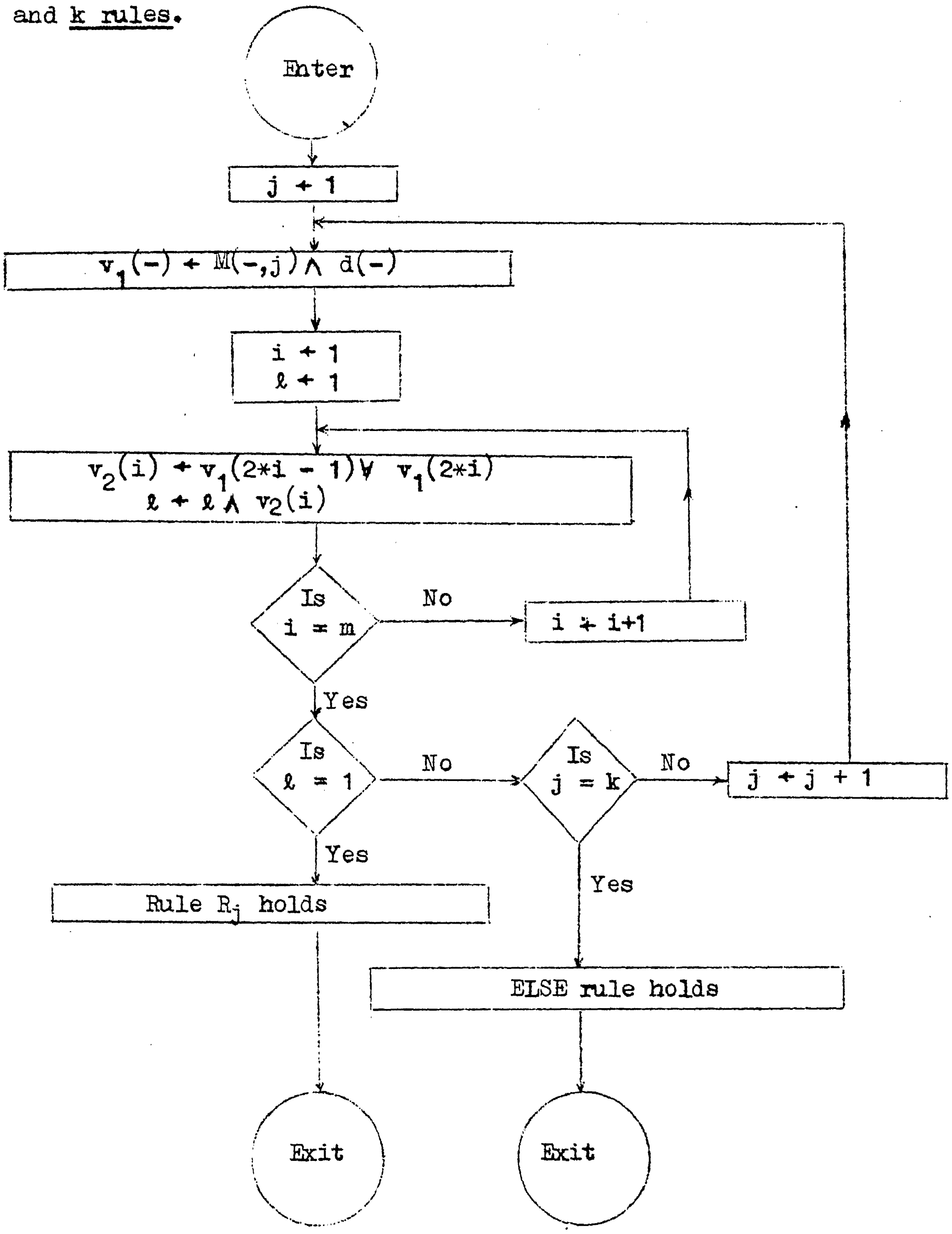
For illustration let us consider the example of algorithm (11).



So $\ell = 0$

We see $\ell = 0$ and so conclude rule R1 does not apply.

The flow chart for the step 3 of this algorithm is presented in Figure 4.7. It is assumed that the decision table contains m conditions and k rules.



Flow Chart for Step 3 of the Modified Algorithm

Figure 4.7

CHAPTER V

SKETCH OF PROGRAMMABLE LOGIC CONTROLLER

The programmable logic controller (PLC) is to be designed to execute decision tables. Execution of decision tables involves evaluation of data vector d from a set of condition addresses and then performing step 3 mentioned in the modified algorithm presented in the previous chapter. Inside PLC each condition will be assigned an address and the address will uniquely identify whether a condition is an input line, or an output line, or an internal function, or an internal binary variable. Actions will also be coded as concatenation of op code which determines the type of action and address of the variable to be acted upon. As address part of the instruction identifies the type of variables, the total number of instructions needed are less than the total number of actions. For the listing of instructions and the actions they represent see Figure 6.3.

5.1 DATA STRUCTURE

Timers :

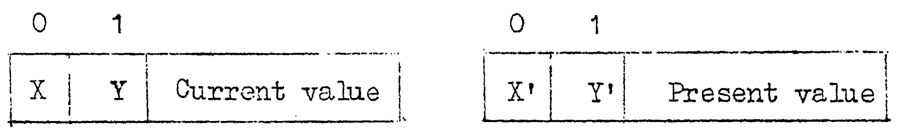
Industrial control applications invariably require a number of timers. The number of timers required depends upon the application. But to be on the safe side a large number of timers should be available to users.Timers are real time devices and times events. It is also to be noted that more than one timer can be 'ON' at a moment or during an interval of time. There are two ways to implement this device.

Method 1

We can have a number of hardware counter units. Whenever a timer is to be started, the counter corresponding to it can be enabled by setting a flip-flop. All the counters can be driven by a single clock generator. The clock generator can be obtained by feeding master clock pulses to a suitable scalar. Whenever timer status is required, by comparing timer (counter) unit contents with a present value, one can know if timer is preset value or not $\geq$ the preset value. Present value has to be stored somewhere.

Method 2

This method employs a memory for timers. Two consecutive memory locations are reserved for each timer. First location will store the current value of timer and the second location will store the preset value of the timer. The first two bits have special



First location Next location

significance. Their significance will be clear as we proceed along the chapter. When the scalar sends a pulse all the first locations belonging to active timers are incremented by one. By the word 'active' we mean those timers which are timing the interval. The status of timers can be found by comparing first location with the second location. If we assume scaler sends pulse every 0 1 sec and it takes 10 micro-sec to examine a timer and process it and there are 100 timers then it will take 1 m sec to finish the job of processing all timers. Therefore timers need servicing for 1 msec every 100 msecs. In view of this timing relationship there may be two schemes to service timers.

First scheme is as follows :

As soon as scaler generates a pulse it causes interrupt inside PLC and PLC branches to timer service routine. It processes all the timers and then returns back to its normal routine.

The second scheme is to have a parallel processing unit for timers. It can be assigned the responsibility of incrementing the active timers as soon as scaler pulses arrive. The benefits derived from it would be limited if it does only this much. We can assign it the responsibility of 'starting' a timer (or equivalently making it active), supplying the status of a timer as and when demanded by the other processor which may be called now-onward main controller. Inclusion of a separate processor will simplify the control hardware to some extent by dividing the responsibilities. We can further increase the load on the processor for timers by allowing it to handle all the internal functions and internal variables. Let us call this processor by the name Internal Function Processor or IFP in short. IFP can be assigned the duties of incrementing or decrementing a counter, setting or resetting an internal variable etc. In this thesis the second scheme will be used primarily because it simplifies the control hardware and improves the response time of main controller besides other benefits which may be evident as we proceed along the rest of the thesis.

Counter The counter also occupies two consecutive locations in the memory. First location keeps the current count and the second location keeps the preset count. Counters are also stored in the same memory (in which timers are stored). To distinguish timers from counters a bit labelled X or X' is made use off.

0	1	
X	Y	Current Count

0	1	
X'	Y'	Present Count

Bit X or X' = 0 means the location denotes a timer

Bit X or X' = 1 means the location denotes a counter.

The TF processor handles all the problems associated with timers. We can now examine the overall organization of PLC.

5.2 ORGANIZATION OF PLC

We presented a model (cf: Figure 2.15) to represent a decision table control system. Overall organization of PLC falls out from this model.

Internal Function Processor

The block labelled Internal Functions/Variables (cf: Figure 2.15) represents timers, counters and internal variables and can be replaced by Internal Function Processor and its associated memory (IF memory) to store these functions or variables (Figure 5.1). The block can not be just replaced by a memory because these functions require real time processing along with processing of other variables interacting with the system.

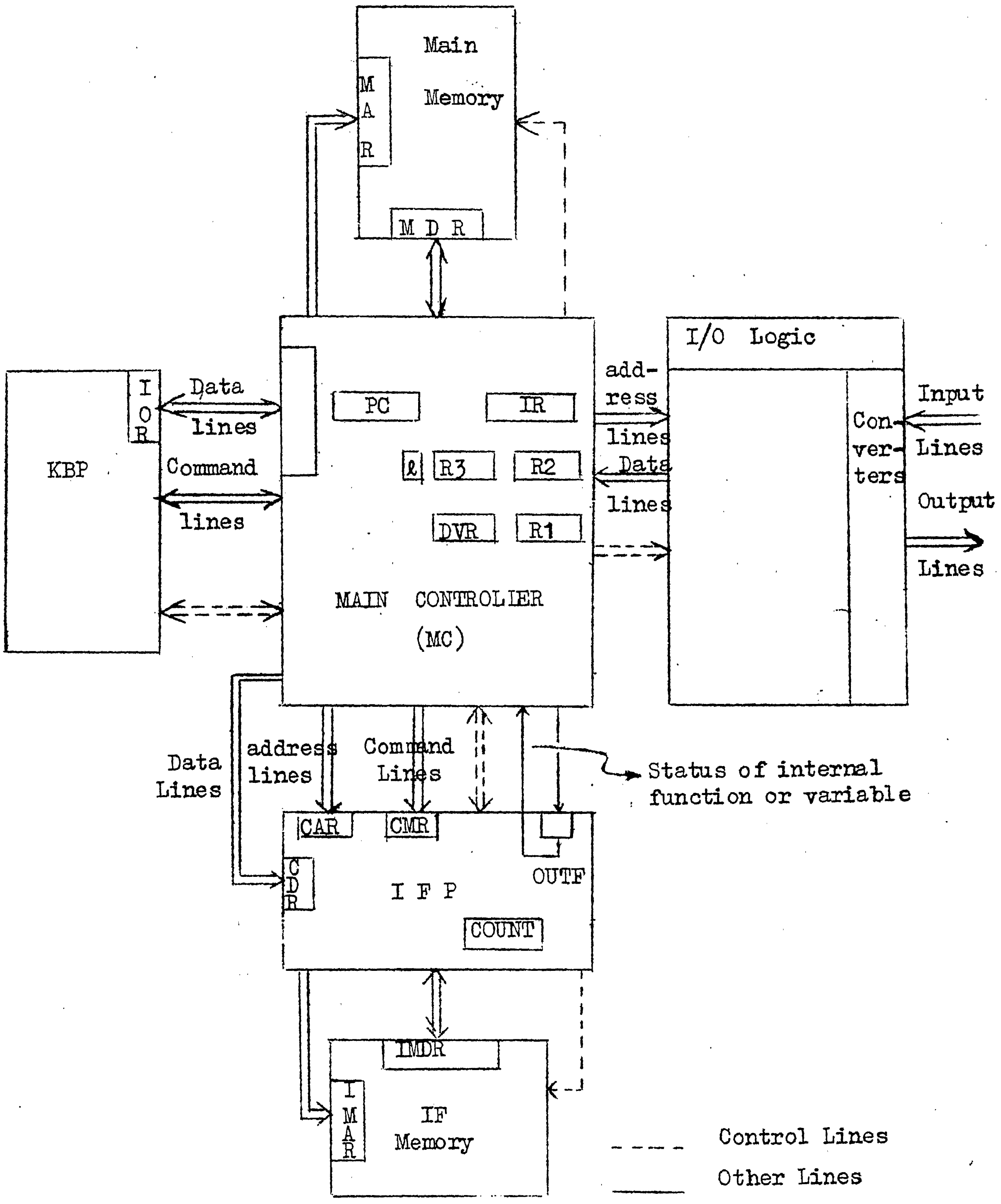
Main Controller :

The block 'Decision Process' (cf : Figure 2.15) can be replaced by a processor which stores the decision tables in a memory and interprets them at real time control of the process. This block has been replaced by its hardware equivalent Main Controller and its memory (Figure 5.1). The main memory stores the decision tables. As 'Decision Process' act on the blocks 'Internal Functions/Internal Variables' and 'Controlled Variables' through solid lines, MC can interact with IFP (via address and command lines) and with output lines (controlled variables) through 'I/O Logic' block.

I/O Logic Block :

In PLC we have combined the blocks 'Manipulated Variables', 'Indicators' and 'Controlled Variables' into the I/O Logic Block. This has been done because the hardware processing requirements for all of them are similar. This processing consists of fetching the status of manipulated variables, indicating contacts (belonging to input lines), and controlled variables belonging to output lines. Functions of I/O logic block are listed below :

1. It records in the output storage register the output states as set by the controller in accordance with the decision tables stored in the main memory.



BLOCK DIAGRAM OF PLC

Figure 5.1

2 It drives the external output converters "on" or "off" as specified in the output storage register. The converters are needed to change the levels and type of voltages. Industrial applications - require voltages 110 or 220 volts AC or DC.

3. It selects addressed input/output line and transfers its status to main controller.

<u>KEY Board Programmer</u>

Key board programmer (KBP) is provided to enter decision tables into main memory. Keys of all types of actions are to be provided. Timers'/counters' preset values can also be entered. Status of any internal variable, preset values, current counts, current values etc. can be displayed. Decision tables can also be displayed.

<u>Registers of Internal Function Processor</u> (IFP)

CAR, CDR, CMR, COUNT are some of the main registers in it.

CAR - (Communication Address Registers).

It stores the address lines' values.

CDR - (Communication Data Register)

It stores the data lines values.

CMR - (Command Register)

Commands from main controller are stored here.

COUNT - (Counter)

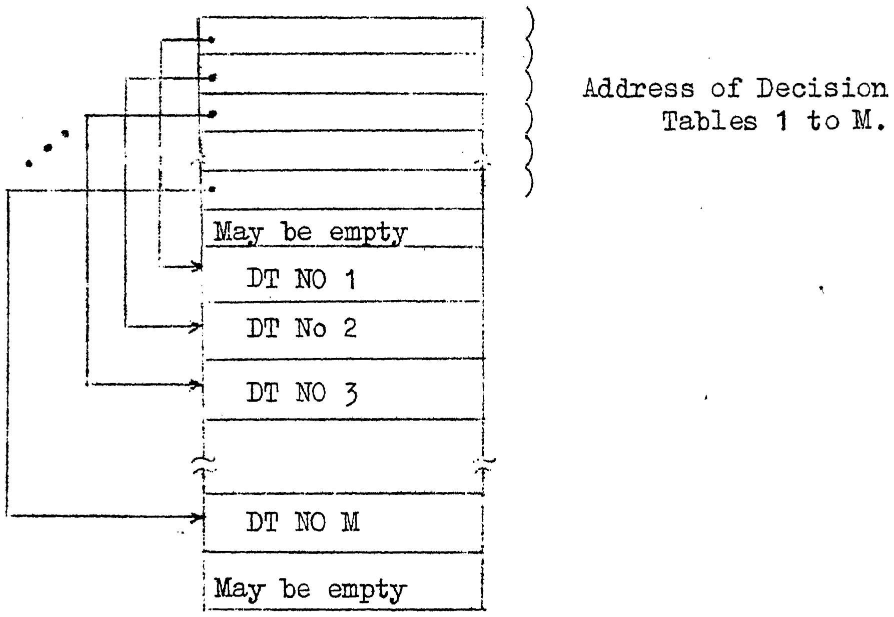
It is a up down counter used to increment timer locations or increment/decrement counter locations.

Number of timers and counters can be exchanged so long as their total do not exceed 128. Address of timer or counter has to be even.

5.3 DECISION TABLE ORGANIZATION IN MAIN MEMORY

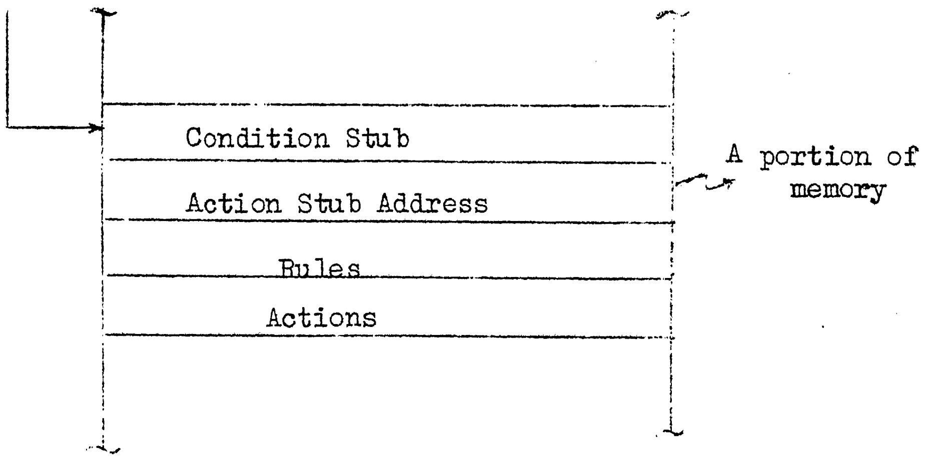
We will store decision tables in their natural sequence. Somewhere in the sequence we might have to skip few decision tables depending upon certain conditions. Note that we have actions 'GDE', 'GDT' . This causes some problems. We have to know the first location from where the said decision table begins. Therefore we need a directory which stores for each decision table, its address also. The simplest scheme to implement this directory is to store address of starting location of a decision table in the location 'DT NO' which is some number and this number uniquely qualifies the decision table. To save memory locations and to avoid some of the problems that otherwise would arise, decision tables are to be assigned consecutive numbers 1,2,3 and so on (Figure 5.2).

Initial few locations are reserved for keeping the beginning address of each decision table. Then each decision table is stored consecutively. In the memory block occupied by a decision table, condition stub is stored first, then a location is kept for keeping the address of action stub, then rules are stored and then action stub follows (Figure 5.3). For us to be able to skip the rules as soon as a rule holds, the action stub address is required.



MAIN MEMORY

Figure 5.2



(Storage organization of a decision table)

Figure 5.3

5.4 SOME OF THE REGISTERS IN MAIN CONTROLLER

1. DVR - (Data Vector Register)

   It is a 14 bit register to store the result of conditions as they are evaluated. It has a shift capability so that loading of two bits per condition can be done from one end.

2. Register R3

   It holds a rule fetched from main memory.

3. ℓ - (link flip-flop)

   It is set or cleared by the output of a combinational logic network whose inputs are registers R3 and DVR. This logic network performs the matching operation and if rule holds, out put of network is 1 else it is 0.

4. Register R2

   The number of current decision table being executed is kept in R2.

5. IR - (Instruction Register)

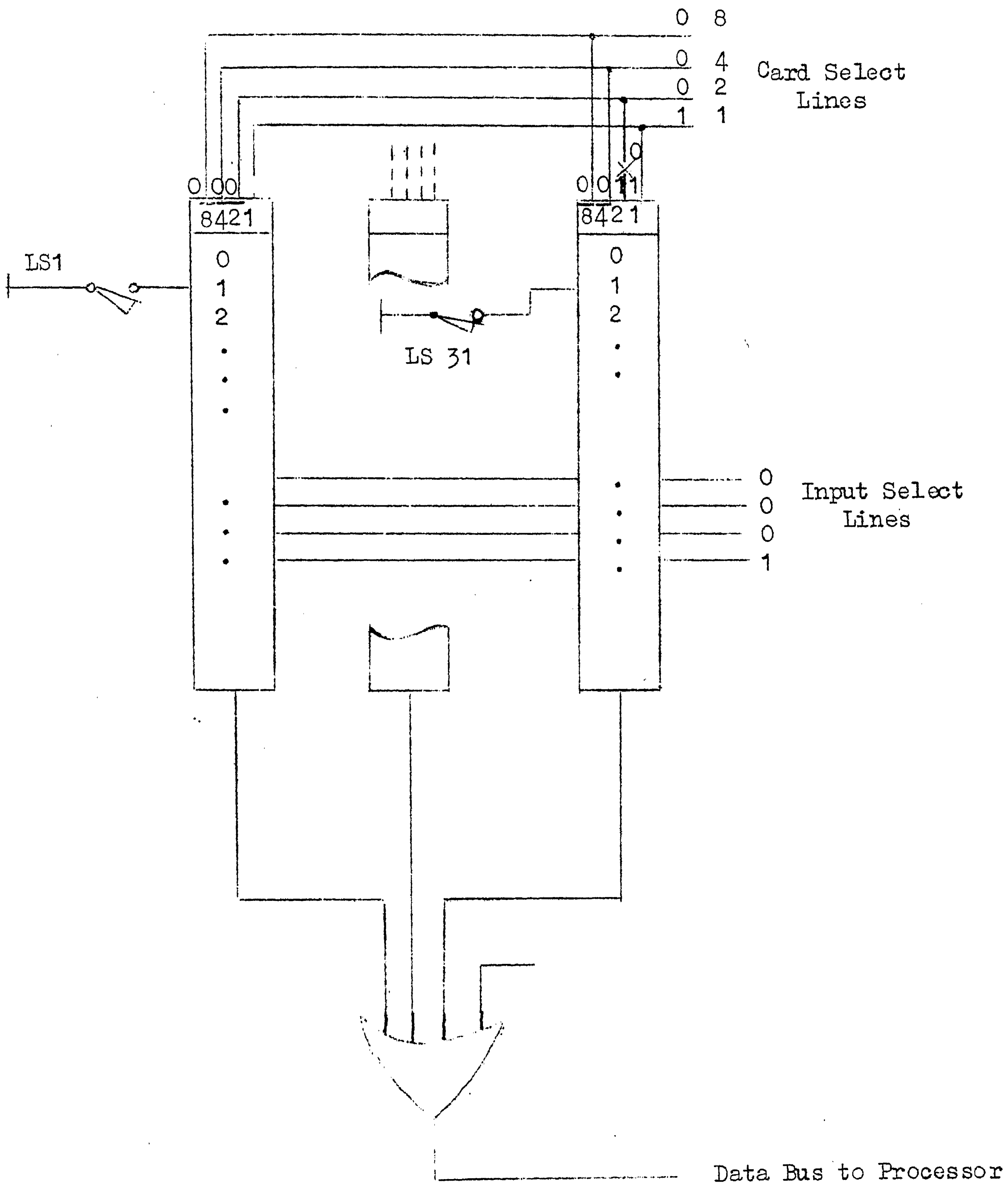
   It holds the instruction to be executed.

6. PC - (Program Counter)

   It contain address of next memory location to be accessed. Next memory location may contain an instruction or a data.

7. DATAR - (Data Register)

   All the words to be transferred to KBP are placed in it. All the words transferred from KBP to main controller are also first placed in DATAR.



0 8
0 4
0 2
1 1
Card Select Lines
0 0 0
8421
0 0 1 1
8421
0
1
2
LS1
LS 31
0
0
0
1
Input Select Lines
Data Bus to Processor

Figure 5.4

5.5 PROGRAMMABLE LOGIC CONTROLLER'S SAFETY

Unlike relay or solid state logic systems, which are comprised of single-purpose or dedicated component, the programmable logic controller shares its logic with all inputs and outputs. This simply means a malfunction of the central processor, memory, or data bus, can affect all outputs. If an adequate detection and shut down system (internal or external to the controller) is not provided, uncontrolled machine motions can occur which would endanger both machines and personnel. This can be shown by study of a typical logic subsystem of programmable logic controller.

Figure 5.4 presents a simplified version of controller's input circuit to demonstrate what can happen to a controller. In the figure, card select lines are in the binary code 'one' to select card no. 1. The input select lines are also selecting input 'one' on whatever card is selected (in this case card 'one'). But because of faulty connection (shown by X) the number three is also being selected. This causes input 'one' on each card to be applied to data bus and we get wrong information 'one' instead of 'zero'. Due to faulty connection, all inputs of both cards will be effectively OR (ED) when the main processor interrogates the number 'one'.

It has been reported by [illegible]. Smith (13) that this type of fault has caused the machine control to go into automatic, start all spindles and take off on its own. The machine cycled with many of the interlocks disabled because the processor wasn't receiving correct information. This problem may be corrected by using parity check technique.

Failure in the memory, the main processor and data bus circuits will cause false commands to be given to outputs. Some mechanism to check operations and shut down the control in the event of failure should be provided.

CHAPTER VI

LOGICAL DESIGN OF PLC

In Chapter V we outlined our PLC. In this chapter we shall define our logic system using a 'hardware description language'. A number of languages are available to describe digital systems and they describe the hardware at different levels such as circuit design, logic design, structure design etc. Their use offers the following advantages (14), among others :

- They serve as a means of communication among computer engineers;
- they permit a precise yet concise description;
- they provide convenient documentation;
- they are amenable to simulation on a computer;
- they aid greatly in an integrated and total design automation - ranging from the design of the computer structure to the wiring list or even to the patterns for making large scale integrated semiconductor devices.

6.1 AHPL

We are going to use 'A Hardware Programming Language' (AHPL) (15) which is due to Hill and Peterson. This language has two types of routines.

- the control sequence routine;
- a combinational logic subroutine.

The control sequence routine is by far the most important. It contains the list of register transfers and branch statements which specify the function of digital system. A translation of this routine will result in a logic block diagram for system hardware. Often the logic functions which must appear in transfer statements become very complex, and therefore impossible to write on a single line. Significant portions of these functions may be replaced by reference to a combinational logic subroutine. A combinational logic subroutine effeciently defines a combinational logic network.

There are five types of statements which appears in control sequence routines. Most frequently used ones are 'branch' and transfer 'statements' (Figure 6.1) .

1. Branch statements
2. Transfer statements
3. Register declaration
4. Bus logic declaration
5. Bus load.

Figure 6.1

Hill and Peterson (15) in his book describes only the first two types of statements and leaves the last three . He feels that they are not necessary as in the discussion of the system he would have described them in detail. He feels, they are required to satisfy the requirements of a possible hardware compiler. However he has introduced some of these statements in the paper published in Computer (14). We will be using them and introduce some more declaration statements (Figure 6.2).

Statement	Meaning
1. INPUTS : f,g, B(4)	f and g are two single bit input lines. B is a 4 bit input line, B_0 through B_3.
2. OUTPUTS : C(8), x,y	C(8) represents an 8 bit out put line, C_0 through C_7. x and y are two single bit out put lines.
3. REGISTERS : PC(13), DVR(14)	PC is a 13 bit register. Bits are represented as PC_0, PC_1,..... to PC_{12}. Similarly DVR is a 14 bit register.
4 FLIPFLOPS : cf, rdycon	cf, rdycon represents two flip-flops named cf and rdycon.

Figure 6.2

Difference between notations

A combinational logic subroutine consists of several type of statements. One of them comes under book keeping statements. In the book (15) he puts B before these statements to identify them.

However we will put instead 'B' to identify them. For numbering the statements Hill and Peterson mostly use numerical numbers. We will mostly use alphanumerical labels. Because of typographical problems we will use M($\perp$ MAR) instead of $M^{\perp MAR}$. Later mnemonics are used to represent particular bit patterns.

6.2. GENERAL INFORMATION

- The word size of both the memories is assumed to be = 16 bits.
- Bit 0 and bit 15 are parity bits.
- Bit 0 is 1 if number of 1's in bits 1 through 14 are even else bit 0 is 0.
- Bit 15 is 1 if number of 1's in bits 5 through 14 are even else it is 0.

  Hence bit 0 is overall parity bit and bit 15 is parity bit for address portion of an instruction or corresponding bits in any other data.

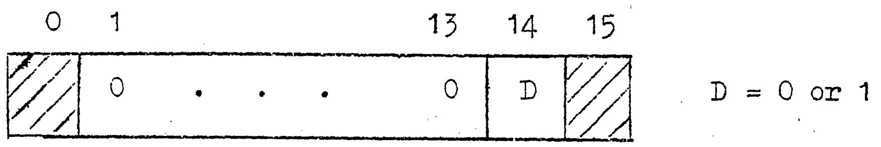
The addresses assigned to variables belonging to different categories are reproduced below :

Addresses	Variable type	Number of variables
0 - 255	Internal binary	256
256 - 510	Timers or Counters	128 (Total)
512 - 767	Output line	256
768 - 1023	Input line	256

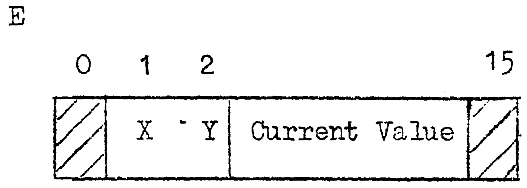
Note that timers and counters can have only even addresses such as 256, 258 etc.

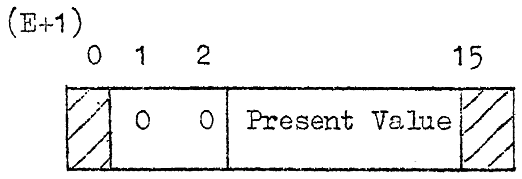
Data structure for Internal Functions:

Internal binary variable : It can take two values 0 or 1. Parity bits 0 and 15 are ignored. Bits 1 through 13 are always zero.



Timer : It occupies two consecutive memory locations of IFP. First location has to be an even address. Parity bits 0, 15 are ignored.





Meaning of the bits marked X,Y is as follows:

X = 0	means	Locations E and (E+1) represent a timer
X,Y = 00	means	Timer is not active.
X,Y = 01	means	Timer is in active state.

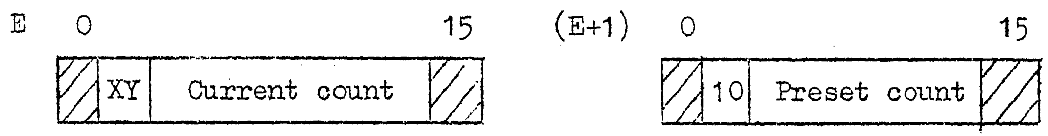
If all the bits 3 to 14 are equal to zero *
then status of timer = 1
else its status = 0

* This dilemma of location 'E being zero' implying status = 1 will be explained shortly.

Meaning of status = 1 is that timer has timed out the interval set on it or timer $\geq$ preset value. Status of 0 means time is $<$ preset value.

Counter :

It also occupies two consecutive memory locations in IFP's memory. First location has to be an even address. Parity bits 0 and 15 are ignored.



Meaning of bits marked X,Y is as follows:

X = 1	means	locations E and (E+1) represent a counter.
X,Y = 10	means	counter can be decremented or incremented. It is in enabled state.
X,Y = 11	means	Counter is in disabled state and can not be incremented or decremented.

The reader may recall that such a restriction has been placed on counter to avoid multiple counting of a single event.

Comment :

Timers and counters deserve one more comment. Let us first take the timer. It is not easy to find out if time is greater than or equal to preset value. This will require an adder or a serial sequential comparator. In the first implementation cost will increase and in the second implementation considerable time will be wasted.

To get over the problem what we can do is the following :-

If asked to clear the timer, load the first location representing the timer by the next location. Note that the second location contains the preset value. When it is required to start the timer just change bit 2 of the first location representing the timer to 1, thereby timer will become active. Now whenever a pulse from the scaler arrives, <u>decrement</u> the first location. Do not decrement if it has already reached a zero value i.e. bits 3 through 14 are zero. It is very easy to test for zero. So if current value contains zero then its status is one or time $\geq$ preset value. This resolves the dilemma mentioned earlier. This method is the faster of the two previous methods suggested and cheaper as well. The same technique may be applied to counters also. Clearing a counter will mean loading the first location representing the counter by the second location. Incrementing the counter will mean decrementing the first location provided the first location (i.e. bits 3 through 14) has not reached zero value. Decrementing the counter will mean, incrementing the first location. Whenever current count reaches zero it means count $\geq$ preset count.

<u>Caution</u> : When the current value reaches zero, counter can not be further incremented and so one should be cautious. It follows that the counter is limited on the positive count side by the preset value and on the negative count side by (2^{12} - 1 - preset value).

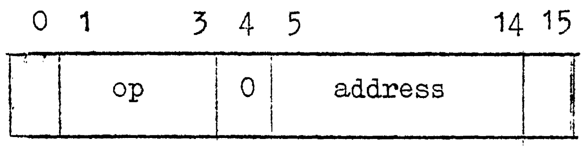
<u>Instructions</u>:

There are seven instructions as shown in the table in Figure 6.3. Because an address uniquely identifies the different types of variables, some of the actions mentioned in Chapter 3 can be represented by a single instruction.

Symbol	Expanded Name	Actions That It Can Represent
SEN	Sense status	It is for condition stub
INC	Increment Counter	A5
DEC	Decrement Counter	A6
CLR	Clear	A2, A4, A7
SET	Set	A1, A3
GDE	Go to Decision Table if ELSE	A8
GDT	Go to Decision Table	A9

Figure 6.3

Instruction format is repeated below :



1. SEN (Sense Status)

It has following functions :

a) Shift data vector register (DVR) left two bits.

b) Load status of addressed variable in the right most two bits of DVR.

If the variable belongs to input out-put line then it is a matter of selecting that line and loading its status. If the variable belongs to internal functions or internal binary variable group then IFP has to be requested for supplying the status.

2. INC (Increment)

   a) Transfer address part of the instruction to IFP on address lines. Note that only the right most 9 bits of address part has to be communicated to IFP.

   b) If any rule other than ELSE is satisfied, ask the IFP to increment the counter. If 'ELSE' rule is satisfied ask the IFP to enable the counter.

3. DEC (Decrement)

   a) Transfer address part of the instruction (only right most 9 bits) to IFP on address lines.

   b) If any rule other than ELSE is satisfied, ask IFP to decrement the counter. If 'ELSE' rule is satisfied ask the IFP to enable the counter.

4. SET

   Case I : NON ELSE Rule is satisfied.

   Its function is dependent upon the address part of the instruction.

A. $0 \leq$ address ≤ 510

   a) Send bits 6 to 14 of the instruction to IFP on address lines.

   b) Ask IFP to set the status of addressed variable. IFP further analyses the address part and its action is as follows:

      (i) $0 \leq$ address < 255 (A1)

      Load the addressed location in memory with 1.

      (ii) $256 \leq$ address ≤ 510 (A3)

Fetch the addressed location. Check if it is timer. If yes then check if it is not active. If so make it active by setting bit 2 to 1. Otherwise instruction is treated as "no operation".

B. $512 \leq$ address ≤ 767.

Set the addressed out put line to 1.

C. $767 \leq$ address ≤ 1023

It is illegal address for this instruction. If this address does occur then it is just ignored and instruction becomes no operation.

Case II : ELSE rule is satisfied.

The steps are exactly same as case I of 'CLR' instruction. 'CLR' instruction follows.

5. CLR (Clear)

Case I : 'NON ELSE' rule is satisfied.

Like instruction SET its function is dependent upon the address part of the instruction.

A. $0 \leq$ address ≤ 510

a) Send 9 bits (6 through 14 of the instruction) to IFP.

b) Ask IFP to clear the addressed variable. IFP further analyses address part and acts as follows:

(i) $0 \leq$ address ≤ 255 (A2)

Loads 0 into its addressed memory location.

(ii) $256 \leq$ address ≤ 510 (A4, A7)

The preset value stored in (address part +1) is loaded into the addressed location. The location (address +1) remains undisturbed. If the addressed location is timer it corresponds to clearing the timer or action A4. If addressed location is a counter this corresponds to clearing the counter or action A7.

B. $512 \leq$ address ≤ 767 (A2).

The addressed output line is cleared to zero. This corresponds to action A2.

C. $768 \leq$ address ≤ 1023.

This is an illegal address for this instruction. But if it does occur then this instruction is ignored and next one in sequence is taken up.

Case II : 'ELSE' rule is satisfied.

The steps are same as Case I of 'SET' instruction.

6. GDE (Go to Decision Table if ELSE rule).

The address part of the instruction is treated as a decision table number. If ELSE rule holds, the register R2 is updated by the address part of the instruction. If a rule other than ELSE holds then nothing is done.

7. GDT (Go to Decision Table)

The address part of the instruction is treated as a decision table number. If ELSE rule holds then nothing is done. If a rule other than ELSE holds, the register R2 is updated by the address part of the instruction.

6.3 SWITCHES AND INDICATORS ON MAIN PANEL

Main controller has following 6 switches.

Switches

1. Power- ON
2. Power- OFF
3. Start
4. Stop
5. Mode Switch
   - (a) Program Entry position (PE)
   - (b) Run Debug position (RD)
   - (c) Run Auto position (RA)
6. CLR-MEM

Functional Description of Switches

Power-ON - It plays a crucial role in start up of the system. It is very essential when PLC is powered, that all the output lines are in low state and necessary registers and flip-flops are initialized to proper initial states. Not initializing the registers (particularly those in system control units) may lead to unpredictable behaviour and may damage machinary. The solution to this problem is to use Power-ON switch contact to trigger one or more monostables and use mono-stable outputs to sequence the power through various sub units of the total PLC. Their outputs can be used to initialize various registers and output lines.

Essentially all this means that Power-ON switch turns on the power in proper sequence to minimize the transients and also fixes the state of the digital system. In our case fixing the state means that first step of hardware program is initiated.

Power-OFF - It will smoothly turn off the power from the system. If a core memory is used, its content will not be lost. If a semiconductor memory (RAM) is used back up battery power will hold the contents of memory.

Start - It is a push button. On pressing it, 'startf' flip-flop gets set. PLC begins execution depending upon mode switch position.

Stop - It is used to halt the operation of PLC. Status of all lines (output) and internal variables or functions remains same. Timer clock cease to operate. 'Startf' flip-flop gets cleared and 'stopf' flip-flop gets set.

CLR-MEM - This is a push button switch used to clear all the timers, counters and internal variables before starting the PLC. For this switch to have effect on PLC it is necessary that startf flip-flop is in zero state. When CLR-MEM switch is down (or pressed) then CLR-MEM line goes high.

Mode Switch -

(a) Program Entry Mode- (PE)

When mode switch is in this position line labelled PE is at "one" logic level. All the changes in program or internal functions or internal variables can be done in this mode. Execution of any decision table is not possible.

(b) (b) Run Debug Mode (RD) -

When mode switch is in this position, RD line is at logical "1" level. Control program i.e. decision tables can not be changed or displayed. However content of internal functions (only preset values) and internal variables can be changed. Any location of IFP's memory can be displayed. Process is controlled in normal manner.

(c) Run Auto Mode (RA) -

When mode switch is at this position, RA line is at logical "1" level. No changes or displays are possible. Key Board Programmer is totally cut off from rest of the PLC. PLC is totally devoted to controlling the process.

Procedure to change modes :-

Once start switch is pressed, mode can not be changed unless stop switch is pressed. Then mode switch can be put at desired position and PLC restarted by pressing the start switch. PLC remembers the position of mode switch at the time of pressing the start switch and ignores position of mode switch later on.

Indicators -

There are five indicators driven by outputs of five flip-flops.

(1) e1 -- error in I/O Logic Block .

(2) e2 -- error in main controller.

(3) pef - PLC is running in program entry mode;

(4) rdf - PLC is running in run- debug mode ;

(5) raf - PLC is running in run-auto mode.

6.4. DESIGN OF START PHASE OF PLC

We are now ready to write the first few steps of AHPL program for PLC. We know that CLR-MEM operates when start switch is still not pressed. It is ineffective once start is pressed. Therefore at step WO, three processes which begin at steps F1.0, W1, S1 are initiated. F1.0 is the first statement of AHPL program for IFP. W1 is the first statement of common program belonging to MC and KBP. S1 is the first statement of the program which handles start and stop switches. At W1 we wait for startf to become one. At W2 we remember the position of mode switch i.e. set flip-flops appropriately. Then we initiate AHPL programs for KBP and MC. KBP's program begins at KO.O and MC's program begins at BO.1.

*Remark : Power-ON switch initiates step WO.

INPUTS : PE, RD, RA, start, stop.

FLIP FLOPS : startf, rdf, raf, pef, stopf

```
WO    DIVERGE (F1.0, W1, S1)
S1    → $((start \wedge \overline{stop}) \times S2 + S4 \times (\overline{start} \wedge stop) + S1 \times (\overline{start \oplus stop}))$
S2    startf ← start
S3    → (S1)
S4    startf ← $\overline{stop}$ ; stopf ← stop
S5    → (S1)
```

W1 $\leftarrow$ (W2 $\times$ startf + W1 $\times$ $\overline{\text{startf}}$)

W2 pef, rdf, raf $\leftarrow$ PE, RD, RA

W3 DIVERGE (KO.O, BO.1)

6.5 INTERNAL FUNCTION PROCESSOR (IFP)

From earlier discussions it is evident that there is a need of communication between main controller and IFP. The kind of communication required is brought out by instructions that we have looked into earlier in this chapter.

It has been mentioned in Chapter V that CMR stores the command from main controller. The bits of CMR are interpreted by IFP. These bits indicate to IFP what to do with various data and address lines. All the necessary commands are listed in the Figure 6.4. 3 bits will be sufficient to describe all the commands.

Mnemonic for Command-code	Meaning	Command may be generated by	
IC1	Supply Status	SEN	instruction/s
IC2	Enable Counter	INC, DEC	"
IC3	Increment Counter	INC	"
IC4	Decrement Counter	DEC	"
IC5	Update Status	SET, CLR	"
IC6	Load Memory Location	*	
IC7	Supply content of addressed location	*	

* These commands are generated by the main controller on a request from Key Board Programmer.

Figure 6.4

Commands IC1, IC2, IC3, and IC4 are self explanatory in view of our earlier discussions. It may not be clear at this stage how update status (IC5) command clears as well as sets the status. A simple trick is employed and will be explained later. IC6 is required to load preset values (counts) in the case of timers (counters). IC7 is required to provide display facility to display preset values (counts) or current values (counts) of timers (counters). IC7 is generated by main controller on demand from KBP.

The various registers, memory, and data paths of IFP are shown in Figure 6.5 . Data path is a 'simple' representation for flow of information from one end to other end. The information in transit may get modified in a predetermined manner depending upon control signals. The meaning and purpose of all the registers are discussed below :

1. CMR (0-2) - Command Register

   It is a 3 bit register to store the commands like IC1, IC2 etc. CMR can be loaded by only main controller and read or decoded by IFP.

2. CAR (0-8) (Communication Address Register)

   Whenever IFP is requested for service and command is supplied to it, command requires a address. The address may be of a variable or a function or may be just referring to a memory location of IFP. The address is placed in CAR by MC.

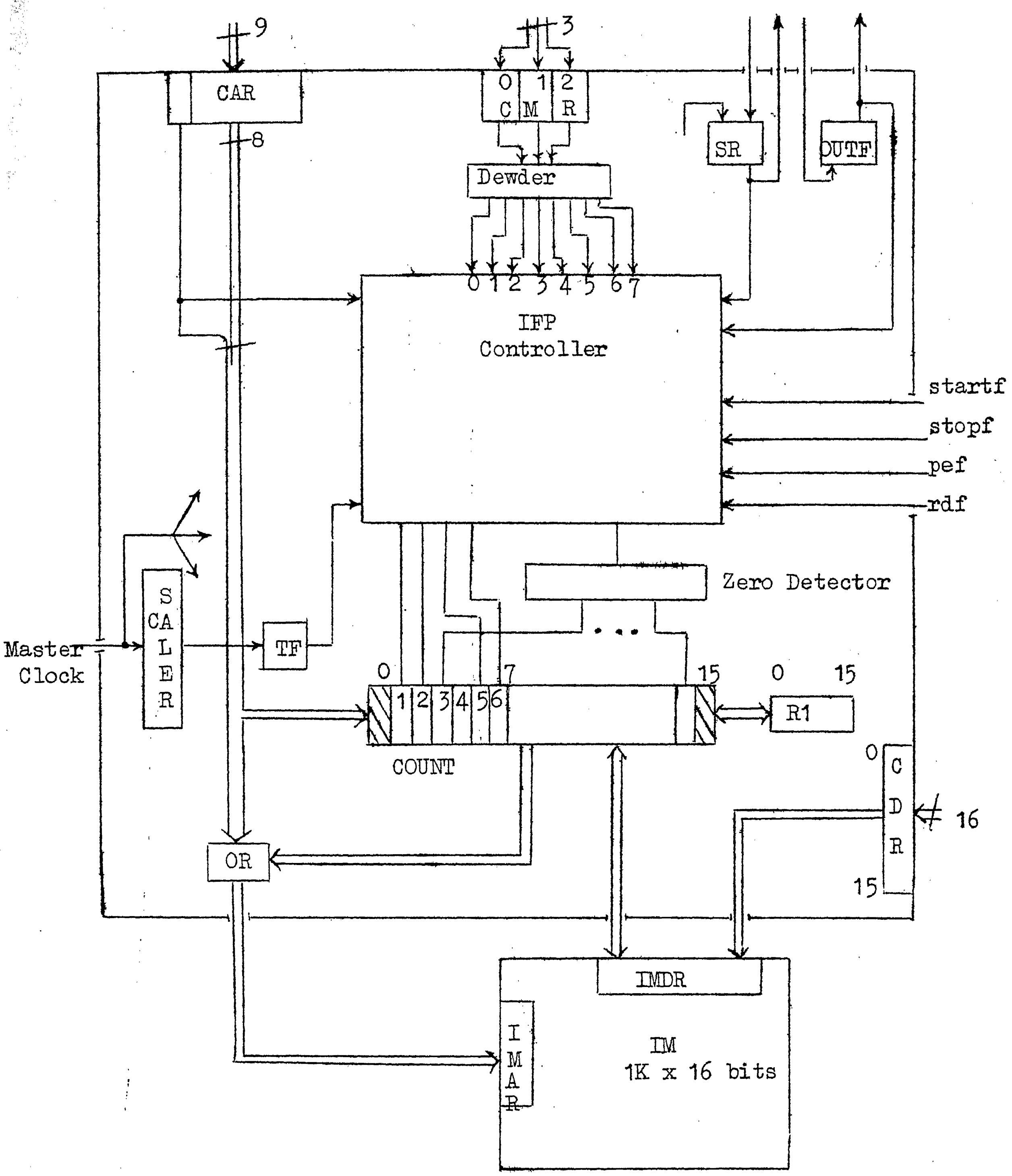

9
CAR
8
3
0 C
1 M
2 R
Dewder
0 1 2 3 4 5 6 7
SR
OUTF
IFP Controller
startf
stopf
pef
rdf
Zero Detector
Master Clock
SCALER
TF
0
7
15
1 2 3 4 5 6
COUNT
0
15
R1
0
CDR
16
15
OR
IMDR
IMAR
IM 1K x 16 bits

Figure 6.5

3. OUTF - (OUT Put Flip- Flop)

It serves two purposes:

(1) In response to the command IC1, IFP supplies status of the addressed variable or function to main controller by setting or resetting OUTF.

(2) OUTF also acts as a extension of the command IC5. If OUTF = 1 then IC5 means setting the variable. If OUTF = 0 then IC5 means clearing the variable. Hence doubts about IC5's abilities to clear as well set the variable are now disposed off.

4. SR - (Service Request Flip-Flops)

When SR = 0 then it means IFP can accept any service request or a command. The main controller can request for service from IFP by flipping the SR from 0 to 1. As soon as SR becomes 1 IFP comes to know a service is required. When IFP has processed the command it responds by clearing the SR. The previous lines bring out following :

- SR = 1 means IFP is busy with previous command.
- SR = 0 means IFP can accept a command.
- MC requests for service by setting SR.
- IFP signals completion of service by clearing SR.

5. IM <0 : 511> (0-15) - (Internal Function Processor's memory)

It is 512 word memory. Each word length is 16 bits. It has to be read and writable memory.

6. IMAR (0-8) - (Internal Memory Address Register)

   It is memory address register capable of addressing 512 locations of the memory IM.

7. IMDR (0-15) - (Internal Memory Data Register)

   It is memory data register. Transfer of information between memory and IFP's register takes place through the register IMDR.

8. COUNT (0-15)

   It is a 16 bit register. Bits 0 and 15 are not used. Bits 3 through 14 can be treated as a up/down counter. It can act as temporary storage register. Bits 1,2,5,6 can be accessed individually for control purpose. Bits 3 through 14 can be checked for all being equal to zero.

9. R1 (0-15)

   It is a temporary register and used only to store intermediate data of COUNT register. Data can be transferred from COUNT to R1 or from R1 to COUNT.

10. SCALER

   It is a scaler register. Essentially it is a counter. It gives a output pulse every 0.1 sec. The input to SCALER is master-clock. Scaling factor depends upon clock frequency employed. It will be assumed that output pulse of SCALER sets a TF flip-flop.

11. TF - (Timer Flip-Flop)

   This flip-flop gets set whenever SCALER gives output pulse. The flip-flop TF is cleared by IFP.

12. TSW - (Timer Switch Flip-flop)

When TF is found to be set, it is cleared and TSW is set. TSW remains set till all the timers have not been processed. When all the timers have been processed TSW is also cleared. TSW is used as a condition to return to timer process-routine from request-service routine.

Logical Design of IFP

First statement of AHPL program of IFP has been labelled F1.0. F1.0 is initiated from the step WO of the program in the last section.

REGISTERS : IMAR (9), IMDR (16), COUNT (16), R1(16), CDR (16), CAR (9), CMR (3)

FLIP FLOPS : SR, OUTF, TSW, TF

INPUTS : CLR MEM

* Remark : F1.0 is a branch statement with a waiting loop. CL1.0 is the first statement of the program which clears internal functions/variables. MO. denotes the main program. A combinational logic subroutine INCT to increment the COUNT and DECT to decreament the count are presented later.

F1.0 $\rightarrow$ (CL1.0 $\times$ $(\overline{\text{startf}} \wedge \text{CLR MEM})$ + MO.2 $\times$ $(\text{startf} \wedge \overline{\text{CLR MEM}})$ + F1.0 $\times$ $(\overline{\text{startf} \oplus \text{CLR MEM}})$)

* Remark : Internal binary variables (0-255) are cleared below :

CL1.0 $\omega^{14}(15)/\alpha^{15}(16)/\text{COUNT} \leftarrow \alpha^{0}(14)$; IMDR $\leftarrow \alpha^{0}(16)$.

CL1.1 IMAR $\leftarrow \omega^{9}(15)/\alpha^{15}(16)/\text{COUNT}$

```
CL1.2     IM (⊥IMAR) ← IMDR ;
          $\omega^{12}(15)/\alpha^{15}(16)/COUNT$ ← INCT(COUNT)

CL1.3   → (CL2.0 × $COUNT_6$ + CL1.1 × $\overline{COUNT_6}$ )
```

* Remark : Internal functions are cleared below. Clearing means loading prese value into current value location.

```
C2.0      R1 ← COUNT ; $\omega^{12}(15)/\alpha^{15}(16)/COUNT$ ← INCT (COUNT)
C2.1      IMAR ← $\omega^{9}(15)/\alpha^{15}(16)/COUNT$; $\omega^{12}/\alpha^{15}/COUNT$ ← INCT(COUNT)
C2.2      IMDR ← IM(⊥ IMAR)
C2.3      IMAR ← $\omega^{9}(15)/\alpha^{15}(16)/R1$
C2.4      IM (⊥ IMAR) ← IMDR
C2.5    →(MO.1 × $COUNT_5$ + C2.0 × $\overline{COUNT_5}$ )
```

* Remark : Combinational logic subroutine INCT follows: Note ($\omega^{0}/COUNT$) represents a null vector. $\wedge/\omega^{0}/COUNT = 1$.

```
    1. CL SUBROUTINE INCT (COUNT)
'B' 2. CTNEW ← $12\rho 0$
'B' 3. i ← 11
    4. $CTNEW_i$ ← $COUNT_{i+3}$ + ( $\wedge/(\omega^{11-i}(15)/\alpha^{15}(16)/COUNT)$)
'B' 5. i ← i-1
    6. i:0 ( ≥, < )   →(4,7)
    7. INCT (COUNT) ← (CTNEW)
    8. RETURN
```

* Remark : Combinational logic subroutine to decrease COUNT follows: Note $\vee/\omega^{0}/COUNT = \vee/\alpha^{0}/COUNT = 0$.

```
    1. CL SUBROUTINE DECT (COUNT)
'B' 2. CTNEW ← $12\rho 0$
```

```
'B' 3.  $i \leftarrow 11$
    4.  $CTNEW_i \leftarrow \overline{(COUNT_{i+3} \oplus (V/(\omega^{11-i}/\alpha^{15}/COUNT))}$
'B' 5.  $i \leftarrow i-1$
'B' 6.  $i : 0 \ (\geq, \ <) \rightarrow (4,7)$
    7.  DECT(COUNT) $\leftarrow$ CTNEW
    8.  RETURN
```

* Remark : MAIN ROUTINE now begins.

Step MO.1 waits for startf to become one. Instant of pressing start switch is treated as zero time. and so SCALER is cleared. TF which records the output pulse of SCALER is cleared to zero to make it ready to receive next pulse. TSW is also cleared. There are two major routines. First is timer process routine beginning at MT1.0. The second is request service routine beginning at MRO.0. Whenever TF gets set, TSW is set and TF is cleared to make it ready for next pulse. TSW remains ON till all the timers have been examined. While IFP is busy in serving timers if a request arrives as evident by SR = 1, the job of timers is suspended and branching takes place to request service routine. See statement MT1.0. (After processing each location MT1.0 is entered.) The request service routine checks TSW and returns control to appropriate place. Both routines are written s.t. in case TSW = 1 and there is piling of service requests timer and service routines are entered alternatively.

```
MO.1    $\rightarrow ((\text{startf} \wedge \overline{\text{CLR MEM}}) \times \text{MO.2} + \text{MO.1} \times (\overline{\text{startf}} \wedge \overline{\text{CLR MEM}}))$
MO.2        $\text{TSW, TF} \leftarrow (0,0)\ ;\ \text{SCALER} \leftarrow \alpha^0$
MO.3    $\rightarrow (\text{MO.3} \times (\overline{\text{TF}} \wedge \overline{\text{SR}} \wedge \text{startf}) + \text{MO.4} \times (\text{TF} \wedge \text{startf}) +$
              $\text{MRO.0} \times (\overline{\text{TF}} \wedge \text{SR} \wedge \text{startf}) + \overline{\text{startf}} \times \text{F1.0})$
MO.4        $\omega^{14}/\alpha^{15}/\text{COUNT} \leftarrow \epsilon^5(14)\ ;\ \text{TSW, TF} \leftarrow (1,0)$
```

*Remark : ϵ is unit vector. $\epsilon^5(14)$ is equal to 2^8. Now timer routine begins. Below a location is read and counter is incremented to next timer location.

```
MT1.0   $\rightarrow (\text{MT1.1} \times \overline{\text{SR}} + \text{MRO.0} \times \text{SR})$
MT1.1       $\text{IMAR} \leftarrow \omega^9(15)/\alpha^{15}/\text{COUNT}\ ;\ \omega^{12}(15)/\alpha^{15}/\text{COUNT} \leftarrow \text{INCT(COUNT)}$
MT1.2       $\text{IMDR} \leftarrow \text{IM}(\perp \text{IMAR});\ \omega^{12}(15)/\alpha^{15}/\text{COUNT} \leftarrow \text{INCT(COUNT)}$
MT1.3       $\text{R1} \leftarrow \text{COUNT};\ \text{COUNT} \leftarrow \text{IMDR}$
```

*Remark : Next step checks bits 1,2 to determine if it is active and non zero timer location. If it is then decrements the location. In case it is not, few steps are skipped.

```
MT1.4   $\rightarrow (\text{MT1.6} \times (\overline{\text{COUNT}_1} \wedge \text{COUNT}_2 \wedge (\perp(\omega^{12}(15)/\alpha^{15}/\text{COUNT}) \neq 0)) +$
              $\text{MT1.12} \times (\text{COUNT}_1 \vee \overline{\text{COUNT}_2} \vee (\perp\omega^{12}(15)/\alpha^{15}/\text{COUNT}) = 0))\ )$
MT1.6       $\omega^{12}/\alpha^{15}/\text{COUNT} \leftarrow \text{DECT (COUNT)}$
MT1.8       $\text{IMDR} \leftarrow \text{COUNT}$
MT1.10      $\text{IM}(\perp \text{IMAR}) \leftarrow \text{IMDR}$
```

*Remark : Saved location value is reloaded into COUNT and then test is made to see if all timer locations are finished. If not finished return to MT1.0. If finished then TSW can be made zero.

```
MT1.12     COUNT ← R1
MT1.14    →($\overline{COUNT_5}$ × MT1.0 + MT1.16 × $COUNT_5$)
MT1.16     TSW ← 0
MT1.18    →(MO.3)
```

*Remark : Command service routines follow. MRO.0 saves COUNT and loads address of the variable into IMAR. IC6 is checked first because it do not require addressed memory location to be fetched. All rest require the addressed memory location to be fetched and loaded into COUNT. Address is available in CAR. In steps MRO.4, MRO.6 memory location is read and loaded into COUNT. IC1, IC2 etc. are to be treated as a <u>3 bit vector</u>.

```
MRO.0     R1, IMAR ← COUNT, CAR
MRO.2    →( (CMR = IC6) × MR6.0 + (CMR ≠ IC6) × MRO.4)
MRO.4     IMDR ← IM (⊥IMAR)
MRO.6     COUNT ← IMDR
MRO.8     CMR : IC1, ( = , ≠ ) → (MR1.0, MRO.9)
MRO.9     CMR : IC2, ( =, ≠ ) → (MR2.0, MRO.10)
MRO.10    CMR : IC3, ( = , ≠ ) → (MR3.0, MRO.11)
MRO.11    CMR : IC4, ( =, ≠ ) → (MR4.0, MRO.12)
MRO.12    CMR : IC5, ( = , ≠ ) → (MR5.0, MRO.1)
MRO.13    CMR : IC6, ( = , ≠ ) → (MR6.0, MR7.0)
```

* Remark : Processing of individual command begins from here. Every command block returns control via the <u>common block M2.0</u>.

IC1 : Command : Supply Status.

```
MR1.0    $\rightarrow (MR1.2 \times \overline{CAR_0} + MR1.6 \times CAR_0)$
MR1.2       $OUTF \leftarrow COUNT_{14}$
MR1.4       $\rightarrow (M2.0)$
MR1.6       $OUTF \leftarrow (COUNT \vee COUNT_2) \quad (\perp(\omega^{12}(15)/\alpha^{15}/COUNT) = 0)$
MR1.8    $\rightarrow (M2.0)$
```

* Remark :

IC2 : Enable Counter

If X,Y = 11 then change X,Y to 10 else do not disturb it.

```
MR2.0 $(\omega^2 / \alpha^3(16)/COUNT) \leftarrow (((COUNT_1 \wedge COUNT_2) \wedge (1,0)) \vee$
                 $((\overline{COUNT_1} \vee \overline{COUNT_2}) \wedge (\omega^2 / \alpha^3 / COUNT)))$
MR2.2       $IMDR \leftarrow COUNT$
MR2.4       $IM(\perp IMAR) \leftarrow IMDR$
MR2.6    $\rightarrow (M2.0)$
```

* Remark :

IC3 : Increment Counter. See INC instruction.

```
MR3.0    $\rightarrow (MR3.1 \times (COUNT_1 \wedge \overline{COUNT_2}) + M2.0 \times (\overline{COUNT_1} \vee COUNT_2))$
MR3.1    $\rightarrow (M2.0 \times (\perp(\omega^{12}/\alpha^{15}/COUNT) = 0) + MR3.2 \times (\perp(\omega^{12}/\alpha^{15}/COUNT) \neq 0))$
MR3.2       $\omega^{12}/\alpha^{15}/COUNT \leftarrow DECT(COUNT) \; ; \; COUNT_2 \leftarrow 1$
MR3.4       $IMDR \leftarrow COUNT$
MR3.6       $IM(\perp IMAR) \leftarrow IMDR$
MR3.8    $\rightarrow (M2.0)$
```

* Remark :

IC4 : Decrement Counter. See DEC instruction.

```
MR4.0    $\rightarrow (MR4.2 \times (COUNT_1 \wedge \overline{COUNT_2}) + M2.0 \times (\overline{COUNT_1} \vee COUNT_2))$
MR4.2    $\omega^{12}/\alpha^{15}/COUNT \leftarrow INCT\ (COUNT);\ COUNT_2 \leftarrow 1$
MR4.4    $IMDR \leftarrow COUNT$
MR4.6    $IM(\perp IMAR) \leftarrow IMDR$
MR4.8    $\rightarrow (M2.0)$
```

* Remark :

IC5 : Update Status ; See CLR and SET instructions. See Figure 6.4.

```
MR5.0    $\rightarrow (\overline{CAR_0} \times MR5.2 + MR5.8 \times (CAR_0 \wedge OUTF) + MR5.18 \times (CAR_0 \wedge \overline{OUTF})$
MR5.2    $IMDR \leftarrow \epsilon^{14}(16)$
MR5.4    $IM(\perp IMAR) \leftarrow IMDR$
MR5.6    $\rightarrow (M2.0)$
```

* Remark : Comes here if timer is to be set i.e. started.

```
MR5.8    $\rightarrow (MR5.10 \times (\overline{COUNT_1} \wedge \overline{COUNT_2}) + M2.0 \times (COUNT_1 \vee COUNT_2))$
MR5.10   $COUNT_2 \leftarrow 1$
MR5.12   $IMDR \leftarrow COUNT$
MR5.14   $IM(\perp IMAR) \leftarrow IMDR$
MR5.16   $\rightarrow (M2.0)$
MR5.18   $COUNT \leftarrow (\alpha^{0}(6),\ CAR, 0)$
MR5.20   $\omega^{12}/\alpha^{15}/COUNT \leftarrow INCT\ (COUNT)$
MR5.22   $IMAR \leftarrow \omega^{9}/\alpha^{15}/COUNT$
MR5.24   $IMDR \leftarrow IM(\perp IMAR)$
```

MR5.26 IMAR ← CAR

MR5.28 IM($\perp$ IMAR) ← IMDR

MR5.30 →(M2.0)

* Remark :

IC6 : Load memory

MR6.0 IMDR ← CDR

MR6.2 IM($\perp$ IMAR) ← IMDR

MR6.4 →(M2.0)

* Remark :

IC7 : Supply content of addressed location.

MR7.0 CDR ← COUNT

MR7.2 →(M2.0)

* Remark :

Command processing ends here. Now the content of COUNT is restored. If TSW = 1 then control is returned to timer process routine.

M2.0 COUNT ← R1 ; SR ← 0

M2.1 →(MT1.1 × TSW + $\overline{\text{TSW}}$ × M0.3)

*** THAT completes the design of Internal Function Processor (IFP) ***

6.6 KEY BOARD PROGRAMMER

This isthe only means provided to enter the control program or set of decision tables into the memory of main controller (MC). Facility to enter the preset quantities of timers and counters is also provided. This is also essential.

The above facilities are a must but many additional features are very much desirable. It is quite likely that a user commits a mistake while entering a data and may or may not know it. Therefore displays are provided. They display all the types of data being entered. A data as it is being entered is simply stored and displayed. When the user is satisfied about the correctness of the data, he presses a special key to terminate the data thence Key Board Programmer (KBP) processes the data.

At initial set up stage of PLC, the control program may have some bugs in it and it may be desirable to examine some decision table stored in the main controller's memory. So the facility to display any decision table is provided. We shall now examine keys, switches and displays on the key board programmer's panel.

6.6.1 Switches

Key Board Programmer (KBP) has two switches S1,S2.

S1 = OFF, S2 = OFF	Means	A whole decision table or part thereof is to be entered.
S1 = OFF, S2 = ON	means	A whole decision table is to be displayed.

S1 = ON , S2 = OFF	means	Internal function's preset values are to be entered.
S1 = ON , S2 = ON	means	Internal functions/variables are to be displayed.

Keys : There are in all 4 types of Keys. Listing of keys can be found in the table in Figure 6.9. Four types are

1. OP Keys,
2. Numerical Keys,
3. LINE Key, and
4. Control Keys.

OP Keys: Their examples are SEN, INC etc. These Keys denotes the op-code of an instruction being entered.

Numerical Keys: Their examples are 0,1,2, - etc. They are used to enter numerical quantities like address part of an instruction or decision table number or rule part of a decision table etc.

LINE Key: This Key is used to generate 0's in op code field of a word. The word with all zeros is treated as a end marker and serves to delimit the condition stub, or rule part or action stub. This is equivalent to drawing a line which separates three parts of our decision table from one another. This end marker may be called Line Marker also.

Control Keys : Their examples are DT, Proceed etc. These Keys tell KBP to interpret the data in a particular way and generate a particular command for MC. Their use will be evident when we go through operating procedures for KBP.

Displays : Following displays are provided.

1. DL - 4 digit decimal display.
2. DR - 5 digit decimal display.
3. TerD - 7 digit ternary (Y,N,-) display.

DL,DR : Display DR isto be located to right of DL. DL usually displays the memory location number where either data is to be entered or from where data is being displayed. First left most digit of DR in case of instructions displays equivalent decimal number representing the op code. Rest four digit displays the address part of the instruction. Note that the maximum value an address part can take is1023. We will provide provisions to have upto 8K = 8192 of main memory (MC's). Hence DL has to be 4 digit.

TerD: This display is used to display a rule part of a decision table while it is being entered or while it is being displayed.

Indicators :

1.	Timer indicator t :	It turns on if timer data is being entered or displayed.
2.	Counter indicator ctr :	It turns on if counter data is being entered or displayed.
3.	Indicator 'go' :	It turns on when the key board is unlocked. As soon as a Key is depressed. 'go' turns off and key board gets locked. As soon as KBP has processed the previous key, go is made on and key board is unlocked.

4. Indicator 'e'	:	When user commits procedural mistakes this error indicator turns on. It can be turned off by pressing 'reset' key.
5. Indicator 'pe'	:	When some hardware error occurs either in transmission or inside KBP then this error indicator turns on. It also can be reset by pressing 'reset' key. Only those hardware errors which result in wrong parity bit/bits are detected.

6.6.2 Procedures to Operate Key Board Programmer

For KBP to be operative it is necessary that power has been turned on by pressing POWER-ON switch on the panel of MC and start has been pressed. In addition MC should be either in PE or in RD mode.

Procedure One*: (To enter decision tables)

*Comment : Both the switches S1 and S2 should be OFF and MC in PE mode.

Step O is required when one is inputting the first decision table. It is explained in chapter 5 that decision tables are stored one after the another in consecutive blocks of memory locations. The beginning address of the first decision table has to be supplied by the user. This purpose is served by step O. Whenever user wants to leave some space between decision tables he has to execute step O.

---

* Readers may refer to the 'Decision Table Organization in Memory' of chapter V.

Step 0 : Press 'entpc' Key. Press numerical keys to enter starting location for the following decision tables. Press the key marked ';' to terminate the number.

	DL				DR				
Displays -	0	0	0	0	0	N1	N2	N3	N4

N1,N2,N3,N4 is the number entered.

Step 1 : Press the key 'DT'. Then enter its number through numerical keys. Then press the key ';'.

	DL				DR				
Display	0	0	0	0	0	N1	N2	N3	N4

N1,N2,N3,N4 is the DT number entered.

Step 2 : For each condition in decision table do the following:

Press the op code key labelled 'SEN'. Then enter address of the variable through numerical keys. When you are satisfied that entries have been correct, press the key; .

	DL				DR				
Display	M1	M2	M3	M4	1	A1	A2	A3	A4

Here M1,M2,M3,M4 is the address of location where condition instruction will be loaded.

Here A1,A2,A3,A4 is address part of the instruction.

Here '1' in DR's first digit is op code for 'SEN'.

Comment - Total number of conditions or condition instructions should be less than or equal to 7. When step 2 is over then to demark it LINE is to be pressed. Hence step 3.

<u>Step 3</u> : Press the key 'LINE'. Press the numerical zero key. Press the key; .

Display

DL				DR				
M1	M2	M3	M4	0	0	0	0	0

Meaning of M1,M2,M3,M4 is same as in step 2.

<u>Step 4</u> : Do the following for each rule in the decision table:-

1. Press the key marked 'code'
2. For each Y entry in the rule press numerical key '1'.
   For each N entry in the rule press numerical key '0'
   For each - entry in the rule press numerical key '-'
3. When rule ends and you are satisfied that all entries are correct, press the key;.

Display
(Typical for
4 condition DT)

DL				DR				
0	1	2	3	0	0	0	0	0

TerD

-	-	-	Y	N	-	Y

Unfilled enteries are filled with -s.

<u>Step 5</u> : Press the key LINE. Press the numeral 0. Press the key ;.

Display - Same as step 3.

<u>Step 6</u> : Do the following for each action in the decision table.

1. Press appropriate action op code key.
2. Enter the address of the variable through numerical keys.

3. When you are satisfied about the correctness of enteries press the key ;.

   Display - Similar to step 2. First digit of DR will contain appropriate code corresponding to the op code. See Figure 6.6 ahead.

<u>Step 7</u> : Press the key 'LINE'. Press numerical 0 key. Press key ;. Display is similar to one in step 3. Go to step 1 if there is any decision table to be entered.

<u>Note</u> -

1. During any step if you detect a mistake in your entries then press the error reset key labelled 'reset'.
2. If any error indicator lamp turns on then press '<u>reset</u>' key and re-enter the data.
3. If you detect an error after the data has been loaded into MC (or equivalently you detect a error after you have pressed the key;) then follow <u>procedure 3</u> to enter any part or word of a decision table.
4. The address of action routine or action stub is entered by main controller itself and therefore you will observe an increment of two in the content of display DL when you move from step 3 to step 4. Whereas usual increment is one.

<u>Procedure</u> 2: (To enter internal functions)

* Comment - Only preset values or preset counts are to be entered.

Note that preset quantities are stored at <u>odd addresses</u>. So <u>add 1</u> to the actual address of timer or counter while entering the <u>address</u>.

Position of switches S1,S2 should be ON, OFF respectively.

Mode of ML = Program Entery or Run Debug.

Step 1 : Press numerical keys to enter the address. When you are satisfied about the correctness of the address, press the key;.

Display

DL				DR				
0	0	0	0	0	A1	A2	A3	A4

A1,A2,A3,A4 is the address entered.

Step 2 : If timer's preset value is to be entered, press the key TMR which is same as LIN. If counter's preset count is to be entered then press the key CTR which is same as SET. Enter preset value or count through numerical keys. When you are satisfied about the correctness of the data, press the key;

Display

DL				DR				
A1	A2	A3	A4	0	V1	V2	V3	V4

where A1,A2,A3,A4 is the address entered in step 1.

V1,V2,V3,V4 is the preset value/count entered in step 2. In case of timers actual value entered should be 10 times the preset value in seconds. This factor of 10 comes because timer least count is 0.1 secs.

Step 3 : Repeat steps 1 and 2 for all timers and counters to be entered.

Procedure 3 : (To alter a part of a decision table)

Note : Number of conditions or number of rules can not be modified. Number of actions can be reduced without any problem. They can be increased if it does not overlay the next decision table. In case number of conditions etc. are to be changed, enter whole decision table using procedure 1. User has to see that the decision table does not overlay the others.

Position of switches - S1 OFF, S2 OFF ;

Mode of MC = Program Entry.

Step 1 : Press the key 'entpc'. Enter the memory address where the data is going to be loaded through numerical keys. Press the key;

Display is same as step 0 of procedure 1.

Step 2 : If entering a condition then execute step 2 of procedure 1.

If entering a rule then execute step 4 of procedure 1.

If entering an action then execute step 6 of procedure 1.

Content of displays will be same as in procedure 1 for corresponding steps.

Following two procedures are for displaying the decision tables or internal quantities.

Procedure 4 : (To display a decision table)

Position of switches - S1 OFF, S2 ON.

MC should in be program entry mode (PE).

Step 1 : Press the key DT. Then enter decision table number through numerical keys. Then press the key;.

Display Sequence -

After the end of step 1, MC begins sending one word after the another till the whole decision table has been sent out. After sending out each word of the decision table, MC waits for the 'proceed' command from KBP. The proceed command is generated when user presses the 'proceed' key. Each word may contain a single condition or a single rule or address of action stub or a single action or the code for line marker that is a string of zeros. For each word its location is also transmitted and displayed on the display DL. The sequence of displays of various quantities like conditions, rules etc. is the same as the sequence in which they are entered in procedure 1. Displays are also the same as in procedure 1. Display of all 0's in DL and DR when action stub is being displayed indicates the end of the decision table.

Procedure 5 : (Display of internal function/variable)

Position of switches : S1 ON , S2 ON

MC should be in PE or Debɡg mode.

Step 1 : Enter the address of the location to be displayed. Press the key;

If $0 \leq$ address ≤ 255 then internal variable is displayed.

DL

A1	A2	A3	A4

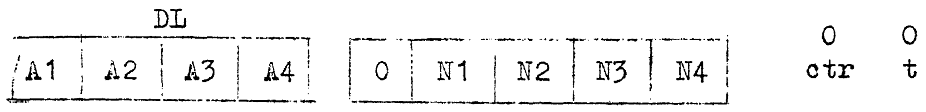
(A1,A2,A3,A4) is variable address

DR

0	0	0	0	D

D is 0 or 1

If address $\geq$ 257 and is odd then preset quantities are displayed.



N1,N2,N3,N4 is preset number

ctr will be 'on' if it represents a counter

t will be 'on' if it represents a timer

If address $\geq$ 256 and is even then current quantities are displayed. Displays are the same as above.

7.6.3 Preparation for Logical Design of Key Board Programmer

Op code Keys represent the op codes of the instructions. The op code keys and corresponding codes are shown in Figure 6.6

Key	Op code bits	Equivalent decimal code
LINE/TMR	000	0
SEN	001	1
INC	010	2
DEC	011	3
SET/CTR	100	4
CLR	101	5
GDE	110	6
GDN	111	7

Figure 6.6

Communication Between KBP and MC

To ensure that correct information is recorded, parity bits 0 and 15 are provided. Whatever message or data is transmitted to MC, MC checks the data or message for validity of parity bits. If an error is found, the KBP is informed so. However KBP does not make any parity checks on messages/datas received by it. In case of error message received from MC, KBP requires the user to notice it and inform KBP by pressing the 'reset' key. In case of no error 'go' light turns on. In case of error, error indicator turns on and after the key 'reset' has been depressed, goes off. We shall call these messages as commands. KBP needs a number of commands to inform MC what type of data is being transmitted and what type of service KBP wants from MC. These commands are listed along with their meaning in Figure 6.7.

We notice that MC also needs to send messages i.e. commands to the KBP. In addition to the commands required to inform KBP about the validity of the messages received by it we need other commands also. They are required in answer to the commands C6 and C7 listed in Figure 6.7. KBP is not going to check for the validity of datas and messages received by it. All the commands from MC to KBP are listed in Figure 6.8.

Note that CR1, CR2 etc. and C1,C2 etc. represent a particular bit pattern. These bits are decoded and then decoder outputs are used to control the operations or flow of information.

S1	S2	Mnemonic for commands	Meaning
OFF	OFF	C1	IOR* contains a number to be entered into PC.
OFF	OFF	C2	IOR* contains a decision table number to be displayed.
OFF	OFF	C3	Data in IOR is to be loaded into M@ (PC) and PC is to be incremented by one.
ON	OFF	C4	Data in IOR is the address of internal function. Possibly save it for use with command C5 .
ON	OFF	C5	Data in IOR is to be loaded into the memory of IFP. The address of the location has been supplied earlier with the command C4 .
OFF	ON	C6	Data in IOR is the number of a decision table to be displayed.
ON	ON	C7	Data in IOR is the address of location in IFP's memory. The content of location isto be displayed.
-	-	C8	You can proceed ahead
-	-	C9	Supply the content of PC.

* IOR is a 16 bit register in KBP. For its description see under the heading "Registers, Flip-Flops and Lines".

@ M is the memory of main controller to store decision tables.

Figure 6.7

Mnemonic for Commands	Meaning
CR1	Last message or data received in parity error, or unexpected data received.
CR2	Last message or data or both O.K.
CR3	Here is a data to be displayed on DL provided S1,S2 $\neq$ (1,1). If (S1,S2) = (1,1) then data is to be displayed on DR.
CR4	Here is condition/action of the DT.
CR5	Here is a rule of the DT
CR6	Here is action stub address

Figure 6.8

Registers, Flip-Flops and Lines

There are in all 25 keys on the key board of KBP. Whenever key board is unlocked and a key is depressed a bit in a register is set. There are two registers Kd and Ky to store their statuses.

Kd (0-10) -	It is a 11 bit register. Bits are set as shown in the table in Figure 6.9.
Ky (0-13) -	It is a 14 bit register. Bits are set as shown in the table in Figure 6.9.

Type of Key	Key Name	Register Bit Affected	Type of Key	Key Name	Register Bit Affected
LIN	LIN/TMR	Ky_o	Control Keys	Proceed	Ky_{12}
Opcode	SEN	Ky_1	"	Reset	Ky_{13}
"	INC	Ky_2	Numerical Keys	0	Kd_0
"	DEC	Ky_3	"	1	Kd_1
"	SET/CTR	Ky_4	"	2	Kd_2
"	CLR	Ky_5	"	3	Kd_3
"	GDE	Ky_6	"	4	Kd_4
"	GDN	Ky_7	"	5	Kd_5
Control Keys	Code	Ky_8	"	6	Kd_6
"	;	Ky_9	"	7	Kd_7
"	entpc	Ky_{10}	"	8	Kd_8
"	DT	Ky_{11}	"	9	Kd_9
			"	-	Kd_{10}

Figure 6.9

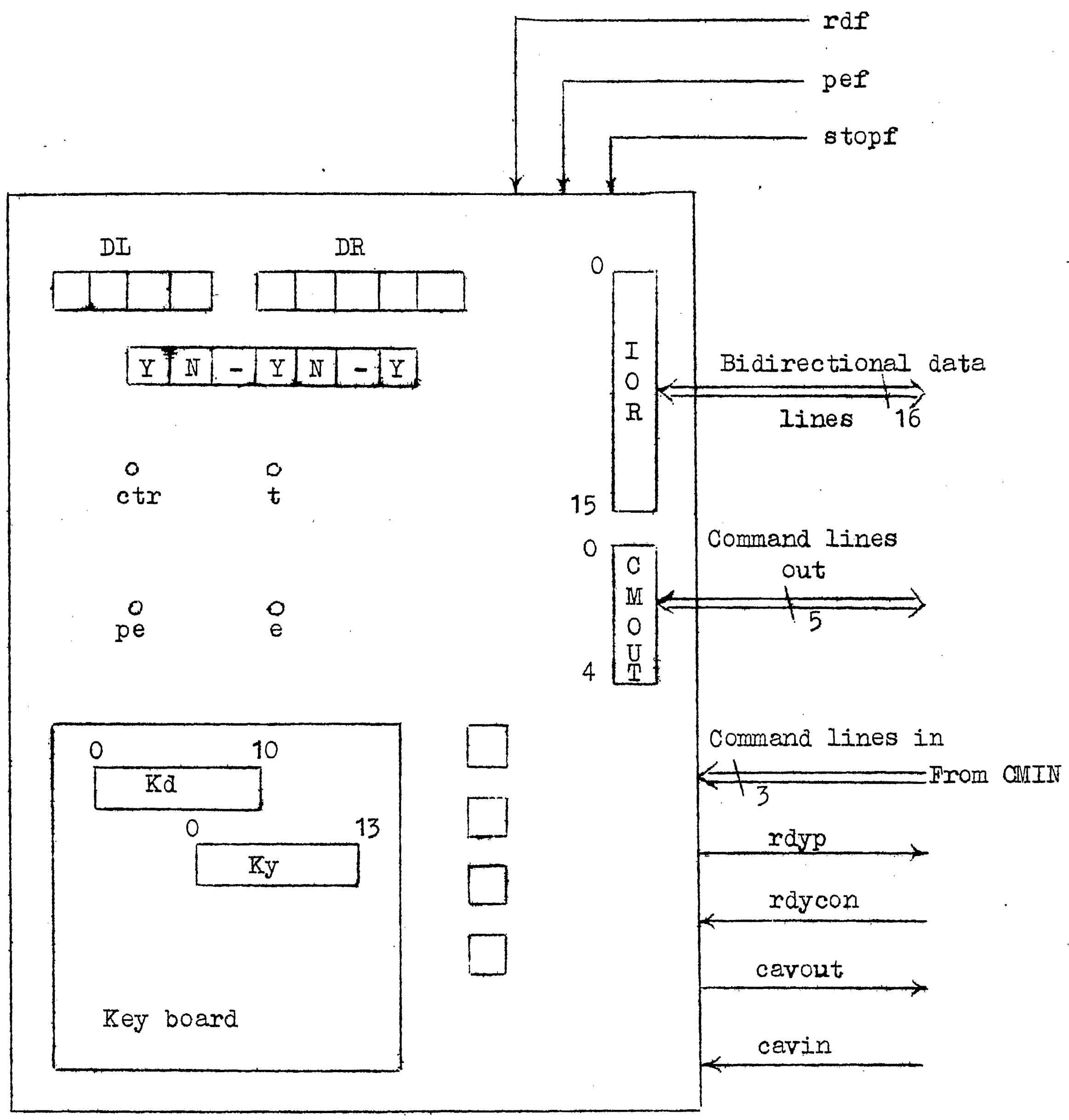

rdf
pef
stopf
DL
DR
Y N - Y N - Y
ctr
t
pe
e
0
I O R
15
Bidirectional data lines
16
0
C M O U T
4
Command lines out
5
Command lines in
From CMIN
3
0
10
Kd
0
13
Ky
Key board
rdyp
rdycon
cavout
cavin

Figure 6.10

Figure 6.10 shows the block diagram of KBP. It shows all the registers to handle data and the lines connecting it to MC.

IOR (0-15) - (Input Output Register)

It is a 16 bit register to transfer a word from KBP to MC or to receive a word from MC. IOR is connected to a 16 bit register DATAR in MC by a bidirectional 16 bit data line.

CMOUT (0-4) - (Command Output Register)

Commands like C1,C2,C3 etc. to be communicated to MC are stored in CMOUT register. There is a 5 bit line provided for MC to sense CMOUT register. Actually 4 bits are enough to code all the 9 commands but a parity bit is also required. Hence we need a 5 bit register.

DL1, DL2, DL3, DL4 - (DL display registers)

Each DLi is a 4 bit register and drives a digit of display DL.

DR1, DR2, DR3, DR4, DR5 - (DR display registers)

Each DRi is a 4 bit register and drives a digit of display DR.

TerD (0-13) - (Ternary display register)

It drives 7 digit ternary display used for rules. Two bit pairs drive a digit of the display. Output of every bit pair can be connected to the display through appropriate combinatorial logic.

rdyp - (ready programmer flip-flop)

When this flip-flop is on it indicates to MC that KBP is free and hence MC can communicate with KBP. rdyp flip-flop is connected to output line rdyp. This is sensed by MC.

rdycon - (ready controller)

It is a line coming from the flip-flop rdycon situated in MC. If this line is 1 then controller can accept a command. rdycon = 1 do not imply that controller is free.

cavout - (Command available line for controller)

When KBP wishes to communicate with MC and finds rdycon = 1 then it pulses the cavout line. This pulse sets cavout flip-flop in MC and tells MC that command is available on CMOUT lines i.e. lines coming from the register CMOUT. In case physical distance between KBP and MC is 'large' then both will be using separate clocks. In this case it is assumed that a pulse synchronizer is there at the MC end.

CMIN (0-2) - (COMMAND IN)

It is a 3 bit line. MC sends commands to KBP on line CMIN. It will be seen (Figure 6.8) that there are a total of 6 commands and so 3 bits are sufficient. Further we do not make any parity checks at KBP's end and thus 3 bits are sufficient.

cavin - Command available on input lines of KBP)

When MC wishes to communicate with KBP and rdyp = 1 then it pulses the cavin line. As soon as cavin goes high KBP comes to know that there is a command on the lines CMIN. Similar to cavout, in case KBP is using a separate clock, pulse synchronizer is needed at KBP-end.

cavinf - (Flip-flop for cavin line)

The flip-flop gets set when cavin line is pulsed.

M - (Memory)

It denotes the memory of main controller MC. It contains 16 bit words. Provision for maximum of up to 8 K memory is made. It is read. writable memory.

pef, rdf - These two lines given the status of the flip-flops pef and rdf talked in sections 6.3 and 6.4.

stopf - This line gives the status of flip-flop stopf in MC.

pe, e, ctr, t - The flip-flops pe, e, ctr, t drive the indicators pe, e, ctr, t.

reset - It is a line inside KBP. If it is pulsed Ky and Kd are cleared and key board gets unlocked.

6.6.4 Logical Design of Key Board Programmer

All the steps which represent Key Board Programmer's hardware begins with K. This however do not hold for subroutines. Two subroutines (control sequence routines) are required. They are 'Binary-to-BCD' and BCD-to-binary conversion routines.

Need for Binary-to-BCD conversion routino:

When we display a decision table or internal functions then we need to convert binary quantities to decimal quantities for easy human comprehension.

Need for BCD-to-Binary conversion routino :

We find it easy to enter all information in decimal numbers and computer finds it easy to manipulate binary numbers and there its need arises.

Need for Register RET (0-2):

We would like to execute above two routines from many places. AHPL does not allow any control sequence routines to be called. Therefore our technique would be to set RET register by a constant and make the last statement of the above routines a jump statement dependent on bits of RET.

Algorithm for binary-to-BCD Conversion -

Consider a register R containing binary number. Consider another register D. Let us divide D into 4 bit decades. Then the procedure is as follows.

Step 0: Clear D to 0.

Step 1: Shift (D,R) 3 bits left.

Step 2: If any decade contains a number greater than 4 add 3 to the decade.

Step 3: Shift (D,R) left 1 bit.

Step 4: Repeat steps 2 and 3 till all the bits in R have been shifted into D.

For details of algorithm refer (16) . The algorithm in AHPL is presented below.

* Remark : Binary number is assumed to be present in bits 2-14 of IOR. BCD number is stored in (DR2, DR3, DR4, DR5) DR is assumed to be zero initially.

```
REGISTERS: IOR(16), DR1(4), DR2(4), DR3(4), DR4(4), DR5(4), CT(4),
           RET(4)
KBD1       IOR ← 2↑IOR
KBD2       DR2,DR3,DR4,DR5,IOR ← 3↑(DR2, DR3, DR4, DR5, IOR)
KBD3       CT ← (0,0,0,0)
KBD4       DR5 ← ADD3(DR5); DR4 ← ADD3(DR4); DR3 ← ADD3(DR3);
           DR2 ← ADD3(DR2)
KBD5       DR2, DR3,DR4,DR5, IOR ← 1↑(DR2,DR3,DR 4,DR5,IOR)
KBD6       CT ← INC(CT)
KBD7       CT:10, (=, ≠) → (KBD8, KBD4)
KBD8       →(KC3.10, KC4.8, KD1.15, KO.3, KO.3) ⊥RET
```

<u>Combinational logic routine</u> <u>ADD3(DR5)</u>

It represents the following truth table. Refer (16).

INPUTS				OUTPUTS			
DR5(0)	DR5(1)	DR5(2)	DR5(3)	A	B	C	D
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0		
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
•	•	•	•	Do NOT care states			
•	•	•	•				
•	•	•	•				
1	1	1	1				

Routine listing follows:

1. CL SUBROUTINE ADD3(DR5)

'B' 2. T ← 9,ρ0

3. $(T1,T2,T3) \leftarrow (DR5_3 \wedge \overline{DR5_1} \wedge \overline{DR5_0}), (\overline{DR5_3} \wedge DR5_2 \wedge DR5_1),$
$(\overline{DR5_3} \wedge DR5_0)$

```
4.  $(T4,T5) \leftarrow (DR5_3 \wedge DR5_2), (DR_2 \wedge \overline{DR}_1)$
5.  $(T6,T7) \leftarrow (DR5_3 \wedge DR5_0), (\overline{DR5}_3 \wedge \overline{DR5}_2 \quad DR5_1)$
6.  $(T8,T9) \leftarrow (DR5_3 \wedge DR5_1), (DR5_2 \wedge DR5_1)$
7.  $ADD3(DR5) \leftarrow (T1 \vee T2 \vee T3), (T4 \vee T5 \vee T3), (T6 \vee T7), (T8 \vee T9 \vee DR5_0)$
8.  RETURN
```

Algorithm for BCD-to-binary conversion

Consider a register D containing BCD number. Consider another register R. Let the converted number to be put in R. The algorithm is as follows.

Step 0 : Clear R to 0.

Step 1 : Shift (D,R) 1 bit right.

Step 2 : Subtract 3 from those decades which contain a number equal to or greater than 8.

Step 3 : Repeat steps 1 and 2 till D contains zero.

We shall then assign routine 'BCD-to-binary' more work. This routine will have to input the whole BCD number then display it, then convert it to binary, then assemble the Op code in DR1 into IOR, then generate parity bits and finally return to appropriate places. Note that maximum BCD number allowed is 8191 = 8K-1 = $2^{13}-1$. Maximum BCD number in an instruction will be $2^{10}-1$ = 1023. Maximum BCD number in case of internal functions will be = $2^{12}-1$ = 4095.

AHPL Routine for BCD-to-binary conversion:

* Remark : DR1 contains Op code in case of instructions. Otherwise DR1 contains zero.

REGISTERS : Kd(11), RG(14), Ky (14)

FLIP FLOPS: ff, go, pb, e, f

```
KDB0   ff ← $\vee/(\text{Ky}, \text{Kd})$
KDB1   → (KDB2 × ff + KDB0 × $\overline{\text{ff}}$ )
KDB2   → (KDB3 × $(\vee/(\alpha^{10}(11)/\text{Kd}))$ + KDB7 × $\overline{(\vee/\alpha^{10}/\text{Kd}))}$ )
```

*Remark: Following 3 steps enter the digit pressed.

```
KDB3   go ← 0;  (DR2, DR3, DR4, DR5) ← $4 \uparrow_{0}$ (DR2, DR3, DR4, DR5)
KDB4   DR5 ← ENDIG(Kd)
KDB5   go ← 1; line reset ← PULSE
KDB6   →(KDB0)
```

*Remark: Test is made for key ; . Then number is converted to binary. K1.0 is error routine's first statement. This will be presented later.

```
KDB7   →($\text{Ky}_9$ × KDB8 + $\overline{\text{Ky}}_9$ × K1.0 )
KDB8   CT ← (0,0,0,0)
KDB9   CT ← INC(CT) ; (DR2, DR3, DR4, DR5, IOR) ← $1 \overset{0}{\downarrow}$ (DR2, DR3,
                                                        DR4, DR5, IOR)
KDB10  DR5 ← SUB3(DR5);
       DR4 ← SUB3(DR4) ; DR3 ← SUB3(DR3) ; DR2 ← SUB3(DR2)
KDB11  CT:14, (= , ≠ ) → (KDB12, KDB9)
```

*Remark: Op code bits are now assembled. They are ORed to IOR bits.

```
KDB12  $(\omega^3/\alpha^4/\text{IOR}) \leftarrow (\omega^3/\alpha^4/\text{IOR}) \vee (\omega^3/\text{DR1})$
```

*Remark: Parity bits 0 and 15 are generated below.

First parity bit 15 isgenerated.

```
KDB13   $RG \leftarrow \omega^{14}/\alpha^{15}/IOR$ ; $pb \leftarrow 1$
KDB14   $CT \leftarrow (0,0,0,0)$
KDB15   $pb \quad RG_{13} \oplus pb$
KDB16   $RG \leftarrow 1 \downarrow RG$ ; $CT \leftarrow INC(CT)$
KDB17   $CT:10, (= , \neq) \rightarrow$ (KDB 18, KDB15)
KDB18   $IOR_{15} \leftarrow pb$
KDB19   $pb \leftarrow RG_{13} \oplus pb$
KDB20   $RG \leftarrow 1 \downarrow RG$; $CT \leftarrow INC(CT)$
KDB21   $CT:14 , (= , \neq) \rightarrow$ (KDB22, KDB19)
KDB22   $IOR_{0} \leftarrow pb$
KDB23   $\rightarrow$ (KC1.5, KC2.5, KC3.14, KD1.5, KI1.4, KI2.9, KF 1.2) $\downarrow$ RET
```

<u>Routine ENDIG</u>

This routine encodes the decimal input lines to BCD digit.

```
    1   CL SUBROUTINE ENDIG (Kd)
'B' 2   $T \leftarrow 4\rho0$
    3   $T_3 \leftarrow Kd_1 \vee Kd_3 \vee Kd_5 \vee Kd_7 \vee Kd_9$
    4   $T_2 \leftarrow Kd_2 \vee Kd_3 \vee Kd_6 \vee Kd_7$
    5   $T_1 \leftarrow Kd_4 \vee Kd_5 \vee Kd_6 \vee Kd_7$
    6   $T_0 \leftarrow Kd_8 \vee Kd_9$
    7   ENDIG (Kd) $\leftarrow$ T
    8   RETURN
```

<u>Routine INC (CT)</u>

Very similar to one presented in IFP design. Change dimension of CTNEW to 4 and in step 3 change i to 3.

<u>Routine SUB3</u>(A)

Let A be 4 bit register (A0, A1, A2, A3). The truth table representing the SUB3 logic is presented below. Refer (16)

	INPUT				OUTPUT			
	A0	A1	A2	A3	B0	B1	B2	B3
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1				
6	0	1	1	0	Don't care states			
7	0	1	1	1				
8	1	0	0	0	0	1	0	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	1	0	0	1
13	1	1	0	1				
14	1	1	1	0	Don't care states			
15	1	1	1	1				

1. CL SUBROUTINE SUB3(A)

'B' 2. $T \leftarrow 4\rho 0$

3. $T_0 \leftarrow A_3 \oplus A_0$

4. $T_1, T_2, T_2 \leftarrow (\bar{A}_3 \wedge A_2), (A_2 \wedge \bar{A}_0), (A_3 \wedge \bar{A}_2)$

5. $T_4, T_5 \leftarrow (A_1 \wedge \bar{A}_0), (\bar{A}_1 \wedge A_0)$

6. $T_6, T_7 \leftarrow (A_1 \wedge A_o), (A_3 \wedge A_2)$

7. $SUB3(A) \leftarrow (T_6 \vee (T_7 \wedge A_o)), ((T_4 \vee (T_3 \wedge A_o)) \vee (\bar{A}_3 \wedge T_5)), ((T_1 \vee T_2) \vee (T_3 \wedge A_o)), (T_o)$

8. RETURN

<u>Main Key Board Routine</u>

*Remark: Main key board routine now follows. KO.O is the step initiated by step W2 of the AHPL routine presented in sec. 6.4. It is necessary, for KBP to operate, that MC is in mode PE or RD. It is checked in the step KO.O. Then indicators and registers are initialized. Then position of switch S1 and S2 is checked and branching takes place to appropriate subportion of the program. When stop is pressed stop f becomes 1. We branch to W4. The program beginning at W4 is presented in sec. 6.8

REGISTERS: DL1(4), DL2(4), DL3(4), DL4(4), TerD (14), RET (3)

FLIP FLOPS: pe, e, t, ctr, cavinf, rdyp, ff, f

INPUTS : S1, S2

KO.O $\rightarrow (KO.1 \times ((pef \vee rdf) \wedge \overline{stopf}) + KO.0 \times (\overline{(pef \wedge rdf)} \wedge \overline{stopf}) + stopf \times W4)$

KO.1 $DL1, DL2, DL3, DL4 \leftarrow \alpha^{o}(16); DR1, DR2, DR3, DR4, DR5 \leftarrow \alpha^{o}(20);$
$TerD \leftarrow \alpha^{14}(14)$

KO.2 $rdyp, e, pe, t, ctr \leftarrow (1,0,0,0,0)$

*Remark: Recall that $Y \rightarrow \begin{smallmatrix}1\\o\end{smallmatrix}$, $N \rightarrow \begin{smallmatrix}o\\1\end{smallmatrix}$, $- \rightarrow \begin{smallmatrix}1\\1\end{smallmatrix}$ hence TerD is initialized to vector of 1's.

```
KO.3   → ($\overline{\text{stopf}}$ × KO.4  +  W4 × stopf)
KO.4   → (KO.3 × $\overline{(\vee/(\text{Ky, Kd}))}$  +  KO.5 × ($\vee$/Ky,Kd))
KO.5     rdyp, cavinf, go ← (0,0,0)
KO.6     (S1, S2): 1, (< , =,> ) → (KC1.0, KD1.0, KO.7)
KO.7     (S1,S2) : 2, ( =, > ) → (KI1.0, KF1.0)
```

*Remark : DECISION TABLES ARE ENTERED HERE.

Comes here if S1,S2 are OFF.. So KBP has to process procedures 1 or 3. Both are same as implementation of them goes. See Figure 6.7. Steps KC1.0 to KC 1.6 process the key entpc = Ky_{10} bit.

```
KC1.0   → (KC1.1 × $Ky_{10}$ + KC2.0 × $\overline{Ky}_{10}$)
KC1.1     line reset ← PULSE ; go ← 1
KC1.2     RET ← (0,0,1)
KC1.3   → (KDBO)
```

*Remark : KDBO is first statement of routine to input decimal number display it and then convert it into binary number.

```
KC1.5     CMOUT ← C1
KC1.6     → (KC5.0)
```

*Remark : Steps beginning with 'KC2 ' handle step 1 of procedure 1. Ky_{11} = DT.

```
KC2.0     → ($Ky_{11}$ × KC2.1 + KC3.0 × $\overline{Ky}_{11}$)
KC2.1      line reset ← PULSE ; go ← 1
KC2.2      RET ← (0,1,0)
KC2.3     → (KDBO)
KC2.5      CMOUT ← C2
KC2.6     → (KC5.0)
```

*Remark : All steps beginning with 'KC3.' process the step 2 and step 6 of the procedure 1. In addition it also generates a word for demarker LINE. Current memory location has to be displayed. This purpose is served by steps KC3.2 through KC3.10. At step KC3.3 op code is converted to BCD code given in Figure 6.6.

```
KC3.0    $ff \leftarrow \vee/(\alpha^8(14)/Ky)$
KC3.1    $\rightarrow (KC3.2 \times ff + KC4.0 \times \overline{ff})$
KC3.2    CMOUT ← C9
KC3.3    $DR1 \leftarrow (0, (Ky_5 \vee Ky_4 \vee Ky_6 \vee Ky_7), (Ky_2 \vee Ky_3 \vee Ky_6 \vee Ky_7),$
                   $(Ky_1 \vee Ky_3 \vee Ky_5 \vee Ky_7))$
KC3.4    $\rightarrow (rdycon \times KC3.5 + \overline{rdycon} \times KC3.4)$
KC3.5    line cavout ← PULSE ; rdyp ← 1
KC3.6    WAIT FOR cavin PULSE
```

*Remark : We expect input command = CR3. If command in $\neq$ CR3 implies parity error. Steps KC3.8 to KC3.10 display location number. Steps KC3.11 to KC3.15 prepare instruction or word.

```
KC3.7    CMIN: CR3, $(= , \neq) \rightarrow (KC3.8, K2.0)$
KC3.8    RET ← (0,0,1)
KC3.9    →(KBD1)
KC3.10   DL1, DL2, DL3, DL4 ← DR2, DR3, DR4, DR5
KC3.11   RET ← (0,1,1) ; line reset ← PULSE ; go ← 1
KC3.12   →(KDBO)
KC3.14   CMOUT ← C3
KC3.15   →(KC5.0)
```

*Remark : By now all key except Ky_8 = code is left with S1=OFF, S2=0 FF. Following steps process key Ky_8 or <u>step 4</u> of <u>procedure 1</u>.

```
KC4.0    $\rightarrow(KC4.1 \times Ky_8 + K1.0 \times \overline{Ky_8})$
KC4.1      CMOUT $\leftarrow$ C9
KC4.2    $\rightarrow(KC4.3 \times rdycon + KC4.2 \times \overline{rdycon})$
KC4.3      line cavout $\leftarrow$ PULSE ; rdyp $\leftarrow$ 1.
KC4.4      WAIT FOR cavin PULSE
KC4.5      CMIN: CR3, C = , $\neq$ ) $\rightarrow$ (KC4.6, K2.0)
KC4.6      RET $\leftarrow$ (0,1,0)
KC4.7    $\rightarrow$(KBD1)
```

*Remark : Comes below when routine at KBD1 converts binary number in IOR to BCD and places it in DR2, DR3, DR4, DR5.

```
KC4.8      DL1, DL2, DL3, DL4 $\leftarrow$ DR2, DR3, DR4, DR5
KC4.9      CT $\leftarrow$ (0,0,0,0)
KC4.10     DR1, DR2, DR3, DR4, DR5 $\leftarrow \overset{0}{\alpha}(20)$
KC4.11     line reset $\leftarrow$ PULSE ; go $\leftarrow$ 1
```

*Remark : Now rules are to be entered. Ky_9 corresponds to the Key ; . INC is combinational logic routine to increment CT.

```
KC4.12     ff $\leftarrow /((\epsilon^9\overline{(14)/Ky)}, \omega^8/\alpha^{10}/Kd))$  NO DELAY
KC4.13     f $\leftarrow ff \vee Kd_0 \vee Kd_1 \vee Kd_{10} \vee Ky_9$
KC4.14   $\rightarrow (KC4.15 \times (Kd_0 \vee Kd_1 \vee Kd_{10}) + KC4.19 \times Ky_9$
                 $+ \bar{f} \times KC4.12 + ff \times K1.0)$
```

```
KC4.15    $go \leftarrow 0 \; ; \; TerD \leftarrow 2 \uparrow_0 TerD$
KC4.16    $\omega^2/TerD \leftarrow (((\,(1,0) \wedge Kd_1) \vee ((0,1) \wedge Kd_0)) \vee ((1,1) \wedge Kd_{10}))$
KC4.17    $CT \leftarrow INC\ (CT)$
KC4.18    $\perp CT:8, \ (= \, , \neq\,) \rightarrow (K1.0, \ KC4.12)$
```

*Remark : Comes here if rule has been entered. Parity bits 0 and 15 are to be generated. At step KC4.23 bit 15 is set. Then we continue from it ahead to generate bit 0 also. Then we generate command C3.

```
KC4.19    $CT \leftarrow (0,0,0,0) \; ; \; pb \leftarrow 1$
KC4.20    $pb \leftarrow TerD_{13} \oplus pb$
KC4.21    $TerD \leftarrow 1 \downarrow TerD \; ; \; CT \leftarrow INC(CT)$
KC4.22    $\perp CT:10 \, , \ (= \, , \neq\,) \rightarrow (KC4.23, \ KC4.20)$
KC4.23    $\omega^1/IOR \leftarrow pb$
KC4.24    $pb \leftarrow pb \oplus TerD_{13}$
KC4.25    $TerD \leftarrow 1 \downarrow TerD \; ; \; CT \leftarrow INC(CT)$
KC4.26    $\perp CT:14, \ (= \, , \neq\,) \rightarrow (KC4.27, \ KC4.24)$
KC4.27    $CMOUT \leftarrow C3 \; ; \; (\alpha^{15}/IOR) \leftarrow (pb, \ TerD)$
KC4.28    $\rightarrow (KC5.0)$
```

* Remark: INTERNAL FUNCTIONS ARE ENTERED HERE.

Procedure 2 : First block takes care of step 1 of the procedure.

```
KI1.0    $f \leftarrow \vee/(\alpha^{10}/Kd)$
KI1.1    $\rightarrow (KI1.2 \times f + KI2.0 \times \bar{f})$
KI1.2    $RET \leftarrow (1,0,1)$
```

```
KI1.3     → (KDBO)
KI1.4       CMOUT ← C4
KI1.5       line reset ← PULSE ; go ← 1
KI1.6     → (KO.3)
```

*Remark: This block processes step 2 of the <u>procedure</u> 2.

```
KI2.0       f ← $\vee/(Ky_1, ky_2, ky_3, (\omega^9/Ky), Kd_{10})$
KI2.1     →$(Ky_0 \times KI2.2 + Ky_4 \times KI2.4 + f \times K1.0)$
KI2.2       t ← 1 ; DR1 ← (0,0,0,0)
KI2.3     →(KI2.5)
KI2.4       ctr ← 1 ;  DR1 ← (0,1,0,0)
KI2.5       DL1, DL2, DL3, DL4 ← DR2, DR3, DR4, DR5
KI2.6       line reset ← PULSE ; go ← 1
KI2.7       RET ← (1,1,0)
KI2.8     →(KDBO)
KI2.9       CMOUT ← C5 ; t, ctr ← (0,0)
KI2.10    →(KC5.0)
```

*Remark : Steps KC5.0 to KC5.3 are common return steps for almost all key-processing routines above. KC5.1 pulses cavout line as appropriate commands are already loaded by earlier routines. Then it is made sure that messages/datas reach properly.

```
KC5.0    $\rightarrow (KC5.0 \times \overline{rdycon} + rdycon \times KC5.1)$
KC5.1      line cavout $\leftarrow$ PULSE ; rdyp $\leftarrow 1$
KC5.2      WAIT FOR cavin PULSE
KC5.3      $\perp$CMIN: CR2, $(= , \neq) \rightarrow (K3.0, K2.0)$
```

*Remark: Next block is error service routine. If Ky_{13} is 1 then it is that user has noticed the error and pressed reset Key. So simply return after initialization. Otherwise insist on his pressing reset key.

```
K1.0    $\rightarrow (Ky_{13} \times K3.0 + \overline{Ky}_{13} \times K1.1$
K1.1      $e, go \leftarrow (1,1)$
K1.2      linereset $\leftarrow$ PULSE
K1.3    $\rightarrow (K1.4 \times Ky_{13} + K1.1 \times \overline{Ky}_{13})$
K1.4      $e \leftarrow 0$
K1.5    $\rightarrow (K3.0)$
```

Remark: Next block is parity error service routine. Similar to one above .

```
K2.0      $pe, go \leftarrow 1,1$
K2.1      line reset $\leftarrow$ PULSE
K2.2      $\rightarrow (K2.0 \times \overline{Ky}_{13} + K2.3 \times Ky_{13})$
K2.3      $pe \leftarrow 0$
K2.4      $\rightarrow (K3.0)$
```

Remark: K3.0 block starts . DL, DR are initialized .

```
K3.0     line reset ← PULSE ; go ← 1
K3.1     DL1, DL2, DL3, DL4 ← $\overset{o}{\alpha}(16)$ ; DR1, DR2, DR3, DR4, DR5 ← $\overset{o}{\alpha}(20)$
K3.2     →(K0.3)
```

*Remark: <u>Display</u> DECISION TABLE

All the statements beginning with labels KD...represent the program to display decision tables. First few statements enter decision table number and display it. Then the DT number is communicated to MC. Thereafter MC transmits information to KBP in proper sequence with proper commands. These commands are analysed or decoded at steps KD1.9 to KD1.12 and the branching to appropriate program takes place.

```
KD1.0     →($Ky_{11}$ × KD1.1 + $\overline{Ky}_{11}$× KP1.0)
KD1.1       line reset ← PULSE ; go ← 1
KD1.2       RET ← (1,0,0)
KD1.3     →(KDBO)
KD1.5       CMOUT ← C6
KD1.6     →(KD1.7 × rdycon + KD1.6 × $\overline{\text{rdycon}}$ )
KD1.7       line cavout ← PULSE ; rdyp ← 1
KD1.8       WAIT FOR cavin PULSE
```

*Remark: Commands decoded below. CR1, CR2 etc. to be treated as representing a particular bit pattern

```
KD1.9       CMIN:CR3 , (= , ≠ ) → (KD1.13 , KD1.10)
KD1.10      CMIN:CR4 , (= , ≠) → (KD1.20 , KD1.11)
KD1.11      CMIN:CR5 , (= , ≠) → (KD1.24 , KD1.12)
KD1.12      CMIN:CR6  (=  ≠)  →
```

*Remark: Comes here if the word in IOR is address to be displayed on DL. So address in IOR is converted to BCD and loaded into display register. Proceed command is generated for MC and control is transferred back to step KD1.6 to finally transmit the command and wait for next command.

```
KD1.13     RET ← (0,1,1)
KD1.14     → (KBD1)
KD1.15     DL1, DL2, DL3, DL4 ← DR2, DR3, DR4, DR5
KD1.16     DR1, DR2, DR3, DR4, DR5 ← $\alpha^0(20)$
KD1.17     COMOUT ← C8
KD1.18     →(KD1.6)
```

Remark : When IOR contains instructions or LINE marker then we come here. As conversion routine converts bits 1 to 14 hence it is necessary to take out op code and set corresponding bits to zero. Routine to convert binary to BCD for ' RET = 100' returns control to KO.3 where KBP waits for another Key to be pressed. (In this case Proceed Key is required)

```
KD1.20     DR1 ← $(0, \omega^3/\alpha^4/IOR)$
KD1.21     $\omega^3/\alpha^4/IOR \leftarrow (0,0,0)$
KD1.22     RET ← (1,0,0)
KD1.23     →(KBD1)
```

*Remark: Comes here if word or IOR contains a rule.

```
KD1.24      TerD ← $\alpha^{14}/\omega^{15}/IOR$
KD1.25    →(K0.3)
    *Remark : Comes here if IOR contains action stub address
KD1.26      RET ← (1,0,0)
KD1.27      →(KBD1)

    *Remark : KBP comes here if the key pressed is not 'DT' key.
              It may be 'proceed'.  If it is not 'proceed' then
              this is error.

KP1.0     →($Ky_{12}$ × KP1.1  +  $\overline{Ky}_{12}$ ×K1.0)
KP1.1      CMOUT ← C8
KP1.2      IOR ← $\alpha^{0}(16)$
KP1.3      DL1, DL2, DL3, DL4 ← $\alpha^{0}(16)$ ; DR1, DR2, DR3, DR4, DR5
                                                    ← $\alpha^{0}$ (20)
KP1.4     →(KD1.6)

    *Remark: Comes here if INTERNAL FUNCTION/VARIABLE is to be displayed.
KF1.0      RET ← (1,1,1)
KF1.1     →(KDBO)
KF1.2      CMOUT ← C7; (DL1, DL2, DL3, DL4) ← (DR2, DR3, DR4, DR5)
KF1.3     →(KF1.3 × $\overline{\text{rdycon}}$  + KF1.4 × rdycon)
KF1.4      line cavout ← PULSE ; rdyp ← 1
KF1.5      WAIT FOR cavin PULSE
KF1.6      CMIN: CR3, (=, ≠) → (KF1.7, K2.0)
KF1.7      ctr, t ← ($IOR_1$, $\overline{IOR_1}$ )
KF1.8      ($\omega^2/\alpha^3/IOR$) ← (0,0).
KF1.9      RET ← (1,0,1)
KF1.10     → (KBD1)
```

6.7 MAIN CONTROLLER

This is the central controlling unit. It interprets the decision tables and controls the process. I/O logic block, IFP, KBP are built to help it in its duties.

The manner in which it interacts with IFP have been already explained in IFP design and at other places. Its interaction with KBP is also explained in KBP design. While designing the main controller we have to know the meaning of each instruction and commands generated by KBP. We should know what KBP or IFP expects. Therefore we might have to frequently refer to earlier pages. Description of registers and flip-flops now follows :

6.7.1 Description of Registers and Flip-Flops

M - Memory

It is 16 bit word size, read / write memory. Total number of words can be upto 8K.

MDR (0-15) - Memory Data Register

It is a 16 bit register. All the communication with memory M takes place through the register. Any word to be written has to be placed in MDR. Any word read is placed in MDR.

MAR (0-12) - (Memory Address Register)

This register is used to address any memory location of M. Address has to be placed in MAR.

PC (0-12) - (Program Counter)

It keeps the address of next sequential location to be executed or operated upon. In case of jumps or skipping of locations, PC content has to be altered.

R1 (0-12) -(Register R1)

It is updated by the address of action routine (or action stub) just before probing of the rules begins. PC is updated by R1 when a rule is satisfied.

R2 (0-9) - (Register 2)

It holds the current decision table number being executed. Jump to a decision table, simply involves altering R2.

R3 (0-13) - (Register 3)

Every rule of a decision table is brought and put in register R3. Bits 1 to 14 of MDR are loaded into MDR.

DVR (0-13) - Data Vector Register)

This register is capable of being shifted left. For each condition instruction,DVRisshifted left two bits and condition status bits (1,0)or(0,1)are placed in bit-positions 12, 13 of DVR.

ℓ - (Link Flip - Flop)

'Match' is performed between registers R3 and DVR and if 'match' holds i.e. rule is satisfied, ℓ is set to 1 else it is cleared to zero.

IR (0-15) - (Instruction Register)

Any instruction belonging to condition stub or action stub is placed in IR. Line marker is also placed in IR.

DATAR (0-15) - (Data Register)

All the transferance of data between KBP and MC takes place through DATAR. A word to be sent out or brought in is loaded in DATAR. Even the words to be communicated between IFP and KBP pass through DATAR.

IOAR (0-10) - (Input-Output Address Register)

It holds properly coded address of input or output lines.

rdycon - (ready controller flip-flop)

It has been mentioned in section on KBP that rdycon is in state "1" when MC is in a position to accept a command. Otherwise it is in "zero" state.

cavout - (command available on 'out' line of KBP)

When this line is pulsed, cavoutf flip-flop inside MC gets set and it comes to know that KBP is wishing to communicate with it.

OTR (0-1) - (OUT-PUT-REGISTER)

This register is used when setting / clearing a out-put bit i.e. a out-put line. If out-put bit is to be set then OTR = 10 else OTR = 01.

Output - (Output command line)

When this line is pulsed the content of OTR get loaded into selected output line.

e1,e2 - Flip - flops

These are two flip-flops to indicate following errors.

e1 is on if there is an error in I/O block

e2 is on if there is an error in main controller.

Status of e1, e2 is recorded in the binary variable 0 and 1 respectively. Therefore variables 0 and 1 can be used to initiate any procedure.

Note : In case there is a break-down in main controller then none of the indicators might light up. But this situation should not occur.

rdf, raf, pef - These are three flip flops which indicate MC is running in run-debug, or run-auto, or program entry modes respectively. They have been described earlier in section 6.3 and 6.4.

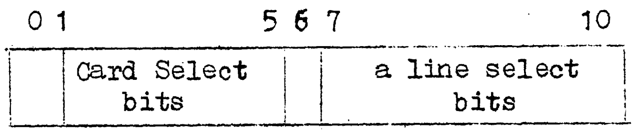
Cavoutf - (Command available on 'out' line of KBP)

This is the flip-flop inside MC, which gets set whenever there is a pulse on the cavout line. Hence cavoutf = 1 means KBP is wishing to communicate with MC.

6.7.2 Input Output Logic Block

This is the block through which PLC communicates with outside world or vice-versa. There are in all 256 input lines and equal number of output lines. It was pointed out in chapter V that input-output addresses should be properly coded so that safety of humans as well as machinery can be ensured to a large extent. Therefore we have decided to provide a 11 bit code of address instead of minimum number 9.

Total number of lines are 512 and all the 512 lines cannot be accomodated in few input or output cards. Moreover selection logic will be enormous if we try to select one line out of 512 directly by decoding a 9 bit or 11 bit address code. We arbitrarily decide to have a total of $32=2^5$ cards. Each card will process 16 lines. Few bits in 11 bit code are reserved to select a card and few bits in the same are reserved to select a line out of 16. To select a card minimum of



IOAR Register

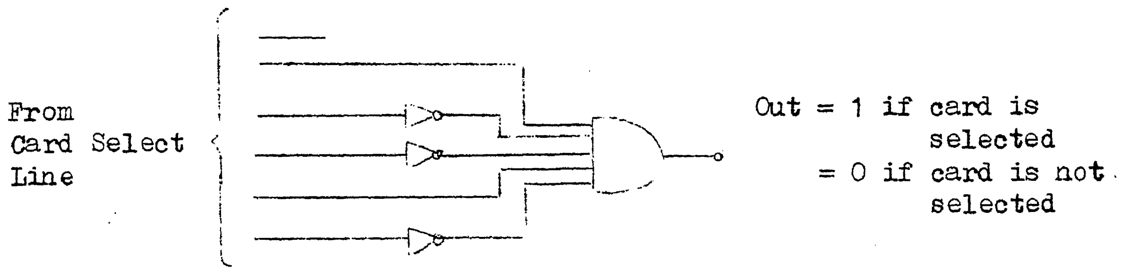
5 bits are required. But 0 is parity but for bits 1 to 5. 4 bits are required to select 1 out of 16. But 6 is parity bit for bits 7 through 10. Odd parity is used. This 11-bit address line is divided into two lines, viz <u>card select line</u> and <u>line select line</u>. The first is 6 bit line and second is 5 bit line.

<u>Design of Input Card</u> -

(1) Each card has an address assigned to it. So a particular line in the card becomes one when that card is addressed. This is explained with an example.

<u>Example</u> :

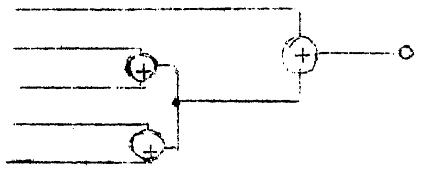
Assume a card has the address $(1\ 0\ 0\ 1\ 0)_2$. Then card select logic (CSLT) will be as follows :



Card Select Logic (CSLT)

(2) Each card has two parity checker units. Output of a parity checker unit is 1 if total number of 1's in the input lines to it, is odd. Parity checker outputs will be used to enable/disable the output bit of the card. 5 bit and 6 bit parity checkers are required.

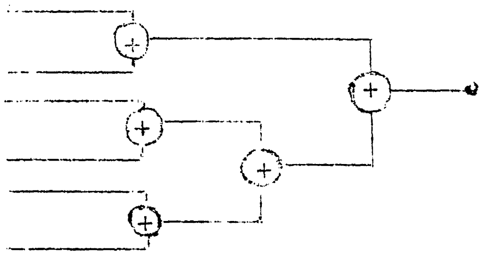
From Line Select Line



out = 1 if no error
= 0 if there is parity error

5 bit parity checker (PCK5)

From Card Select Line



out = 1 if no error
= 0 if there is parity error

6 bit parity checker (PCK6)

(3) Each card has a 4 bit decoder. Input to the decoder is 4 bits of line select line. Parity bit is left out.

(4) Every input line is complemented and then decoder output is used to select one complemented and other uncomplemented line. If output gates (A1, A2, in Figure 6.11) are enabled then the levels of selected lines appear at Q_1' and Q_2' in Figure 6.1). The purpose of having Q_1' and Q_2' is to increase reliability of PLC.

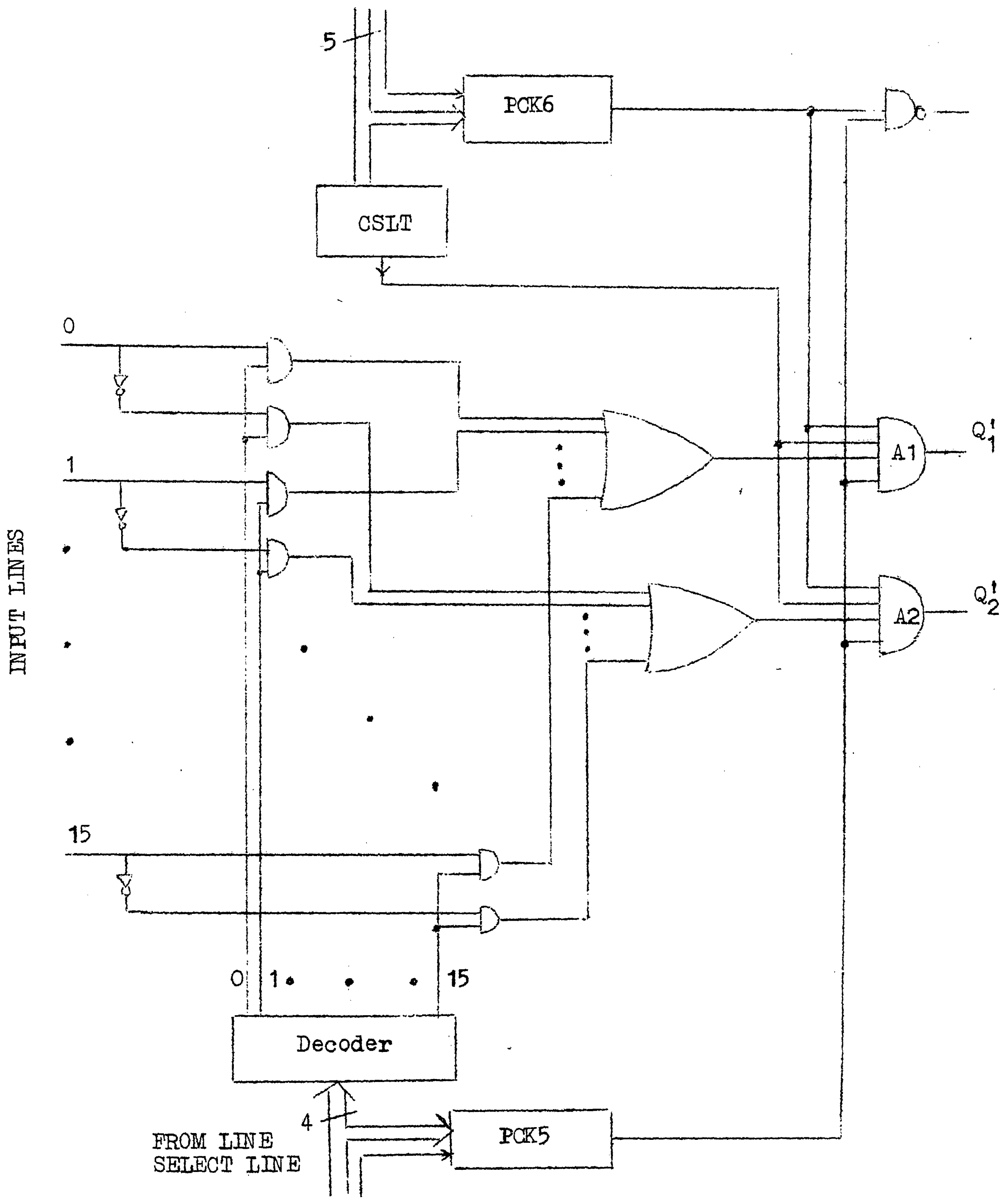

5
PCK6
CSLT
0
1
15
INPUT LINES
A1
Q_1'
A2
Q_2'
0
1
15
Decoder
4
FROM LINE SELECT LINE
PCK5

Figure 6.11

When a selected input line changes its stateit is likely that Q_1' and Q_2' do not change its state simultaneously. Because of spread in device parameter 'delay' and one extra inverter in the path of Q_2', the time a change takes to propagate from input line to Q_2' will be different from that of input line to Q_1'. When any one of Q_1' or Q_2' is changing its state then $Q_1' = Q_2'$. Therefore if at any clock pulse we observe $Q_1' = Q_2'$ then we again check Q_1' and Q_2' at next clock pulse. If this time also $Q_1' = Q_2'$ then there is a problem with the circuitry and we signal a error in I/O block.

We can represent a input card in block diagram form as done in Figure 6.12

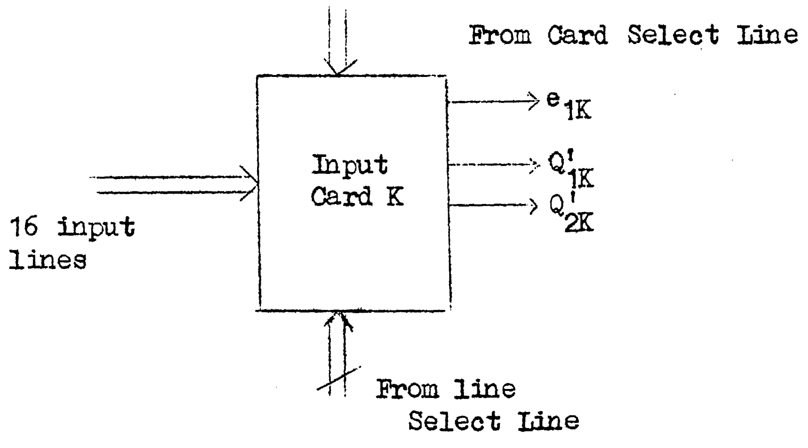


Figure 6.12

Design of OUT PUT Card

Output card is similar to input card. It has a card select unit, parity checkers PCK5, and PCK6 and a decoder. These three units are same as in input logic card. The basic difference comes from a 16 bit register

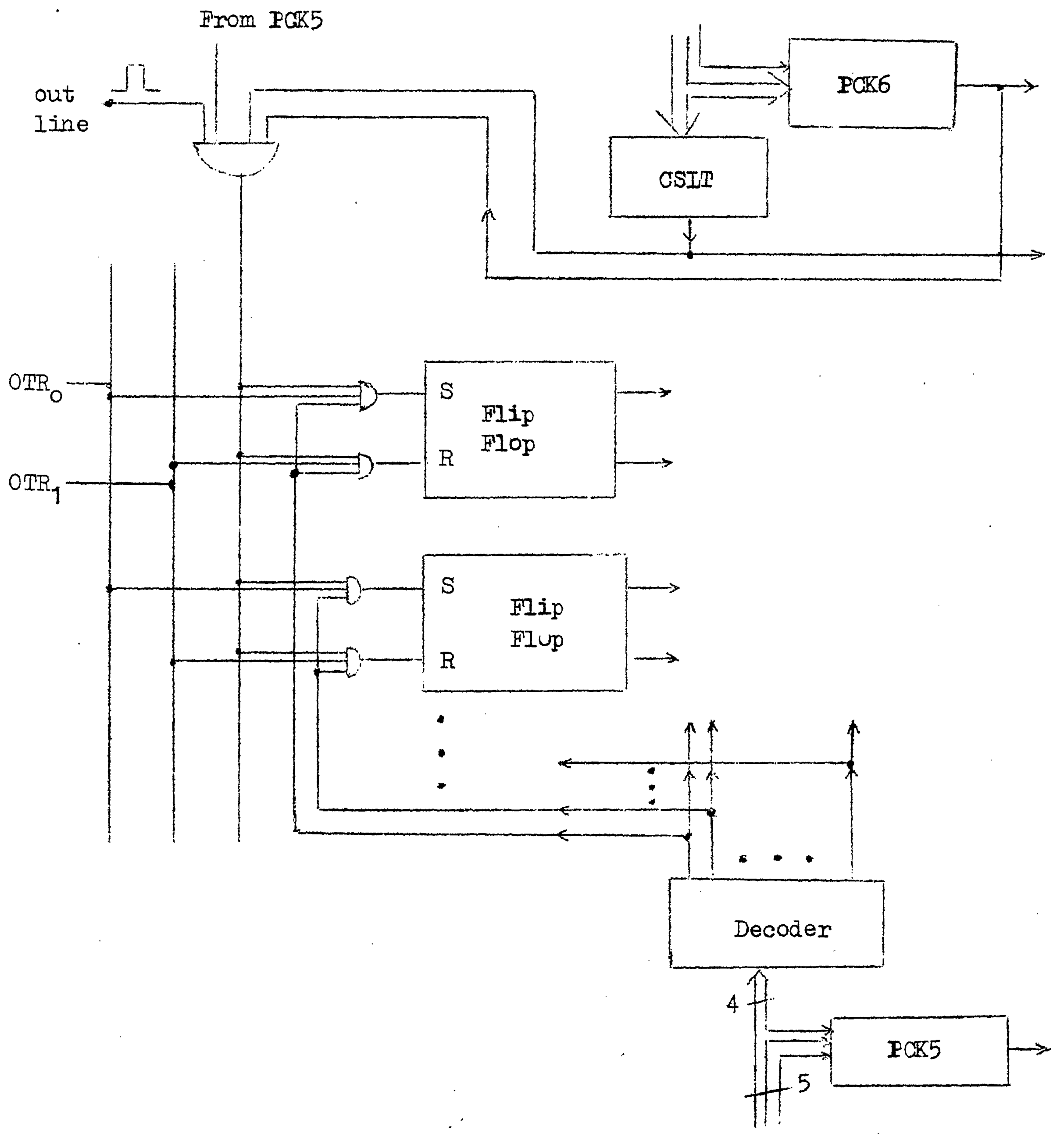

From PCK5
out
line
PCK6
CSLT
OTR0
OTR1
S
Flip
Flop
R
S
Flip
Flop
R
Decoder
4
5
PCK5

Figure 6.13

and associated circuitry to load the register. The register stores the status of output lines. There is a line called 'outline'. When this line is pulsed the data bits at the lines input to the card) are loaded into selected output line. Note that data bits pair (1,0) is for setting the line and data bit pair (0,1) is for resetting the selected output line. The logic which is different from input card logic is shown in Figure 6.13.

Output card can be represented in block-diagram form as shown below in Figure 6.14.

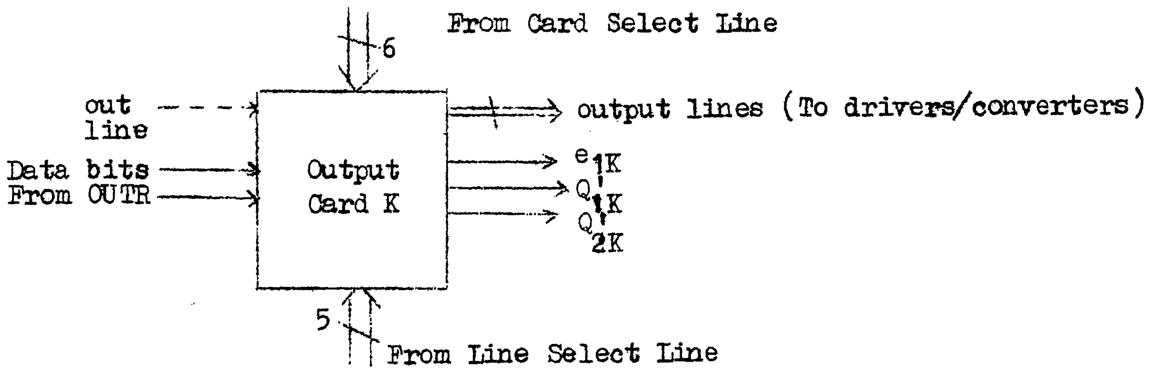


Figure 6.14

Figure 6.14

Let us arbitrarily assume that $card_1$ to $card_{16}$ represent input cards and $card_{16}$ to $card_{32}$ represent output cards. Then the interconnection of input-output cards can be as shown in Figure 6.15. Q_1 and Q_2 are output of selected line and its complement respectively.

<u>Why Do We Have Q_1 and Q_2 Both</u> -

The only reason is safety of human-beings and machinery as pointed out in chapter V. But how do Q_1 and Q_2 help us in achieving our goal ?

Case 1 : Fetching a bits from input or output cards :

If more than one bit are selected due to some error. (Note this error has to be such that error goes unnoticed by parity checkers) then in case all the selected bits have same status we shall have correct output satisfying the condition $Q_1 = \bar{Q}_2$. If any two bits differ we shall have

$$Q_1 = Q_2 = 1$$

and this will immediately warn us. If none of the bits get selected then we shall have

$$Q_1 = Q_2 = 0 \ .$$

Thus $Q_1 = Q_2$ is a signal for error in I/O logic.

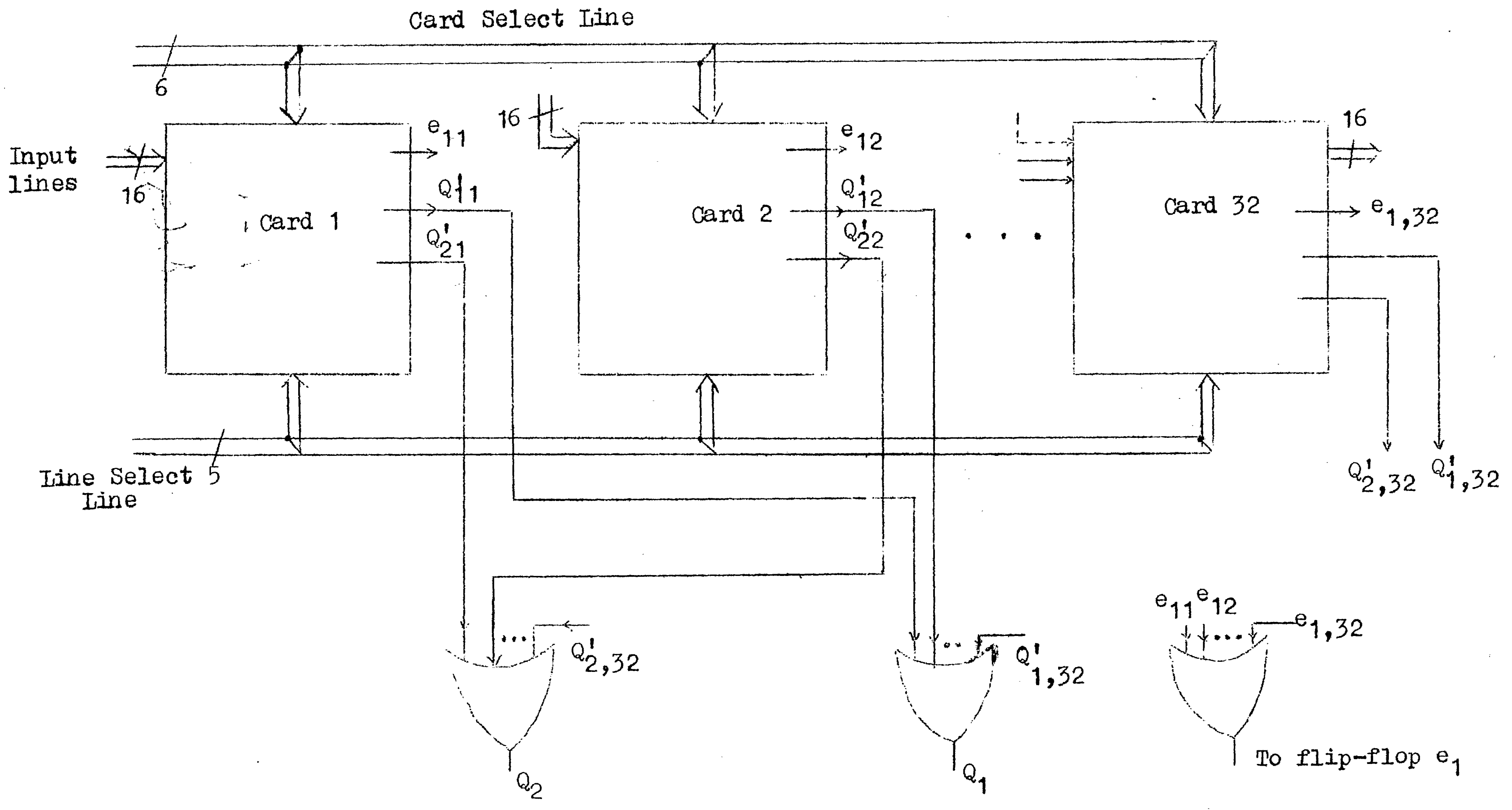

Card Select Line
6
Input lines
16
Card 1
e_{11}
Q'_{11}
Q'_{21}
16
Card 2
e_{12}
Q'_{12}
Q'_{22}
Card 32
16
$e_{1,32}$
$Q'_{2,32}$
$Q'_{1,32}$
Line Select Line
5
$Q'_{2,32}$
Q_2
$Q'_{1,32}$
Q_1
e_{11} e_{12} $e_{1,32}$
To flip-flop e_1

Figure 6.15

Case 2 : Setting a Output bit

Here it helps only in limited way. After setting or resetting an out put bit one can read it again. If it is the expected one then it is alright. But if it differs then we signal the error. A problem still remains. In case more then one bit get selected, we shall alter all the selected lines and not be able to detect it.

6.7.3 AHPL Program for Main Controller

In the following program we have used combinational logic subroutines to select bits Q_1, and Q_2. These routine denotes the combinational hardware presented in back pages to select Q_1 and Q_2. In particular SELECT1 selects Q_1 and SELECT2 selects Q_2. Memory locations 0 and 1 in IFP are reserved for storing status of error indicators e1 and e2. Note e1 and e2 remains set, once they are set. e1 and e2 might be used by user to initiate any shut down sequence.

*Remark :

Declaration statement follows

```
    REGISTERS : R1(13), R2(10), R3(14), DVR(14), PC(13), MAR(13),
                MDR(16), DATAR(16), IR(16), IOAR(11)

    FLIP FLOPS :rdycon, e1, e2, cavoutf, fi, tt
```

B0.1 · rdycon,fi ← (1,0) ; R2 ← $\alpha^0(10)$; e1, e2 ← (0,0)

B0.2 → (B1.1 × $(\overline{\text{stopf}} \wedge \text{raf})$ + D1.1 × $(\overline{\text{stopf}} \wedge \text{rdf})$ + P0.0 × $(\overline{\text{stopf}} \wedge \text{pef})$ + stopf × W4)

*Remark :

Automode program begins at B1.1. Debug mode program begins at D1.1 and program entry mode program begins at P0.0.

Debug mode :

Here we first serve the request if any and then jump to program at B1.1. The last statement of automode program beginning at B1.1 is conditional jump. In case rdf = 1, 'return' is to D1.1 else it is to B1.1. In this way at the most 1 request per execution of a decision table can be disposed off.

Program Entry mode program will be commented later.

D1.1 → (D2.0 × cavoutf + B1.1 × $\overline{\text{cavoutf}}$)

D2.0 rdycon, cavoutf ← (0,0)

*Remark :

PAR1F, PAR2F are combinational function subroutines. Their's output is 1 if there is no error. PAR1F checks parity bit 0 and PAR2F checks parity bit of CMOUT.

D2.1 tt ← PAR1F (DATAR) ∧ PAR2F (CMOUT)

D2.2 → (D2.3 × tt + D5.7 × $\overline{\text{tt}}$)

D2.3 CMOUT : C4 , (= , ≠) → (D4.0, D2.4)

D2.4 CMOUT : C5 , (= , ≠) → (D5.0, D2.5)

D2.5 CMOUT : C7 , (= , ≠) → (D7.0, B1.1)

*Remark : C4 : Refer Figure 6.7

Address in IOR to be saved. We shall save it in DATAR and set fi to 1 to indicate this. We can not save address in CAR as, before C5 arrives, MC might require a service from IFP.

```
D4.0    CMIN ← CR2 ; fi, DATAR ← 1, IOR
D4.1  → (D8.0)
```

*Remark :

C5 : Command to alter internal function. D5.0 checks for availability of IFP. fi = 0 error is signalled

```
D5.0  → (D5.0 × SR + D5.1 × $\overline{SR}$)
D5.1  → (D5.2 × (fi ∧ $DATAR_{14}$) + D5.7 × ($\overline{fi}$ ∨ $\overline{DATAR_{14}}$))
D5.2    fi ← 0 ; CAR ← $\omega^{9}(15)/ \alpha^{15}$/DATAR
D5.3    DATAR ← IOR
D5.4    CDR ← DATAR ; CMR ← IC6 ; SR ← 1
D5.5    CMIN ← CR2
D5.6  → (D8.0)
```

*Remark :

Error Routine follows. Error is signalled to KBP.

```
D5.7    CMIN ← CR1
D5.8  → (D8.0)
```

*Remark :

C7 : Command to display internal function. The return from last statement is <u>conditional</u>. This is because this sub-program is <u>shared</u> by <u>program entry</u> mode program also.

```
D7.0    DATAR ← IOR
DF7.1 → (DF7.1 × SR + DF7.2 × $\overline{SR}$)
DF7.2   CAR ← $\omega^{9}/ \alpha^{15}$/ DATAR ; CMR ← IC7 ; SR ← 1
```

```
DP7.3  →(DP7.3 × SR + DP7.4 × $\overline{SR}$)
DP7.4    DATAR ← CDR ; CMIN ← CR3
DP7.5  →(DP7.6 × rdyp + DP7.5 × $\overline{rdyp}$)
DP7.6    line cavin ← PULSE ; rdycon ← 1
DP7.7  →(rdf × B1.1 + $\overline{rdf}$ × PO.1)
```

*Remark :

Next program finally transmits the command and control is transferred to B1.1.

```
D8.0  →(rdyp × D8.1 + D8.0 × $\overline{rdyp}$)
D8.1    line cavin ← PULSE NO DELAY
D8.2    rdycon ← 1
D8.3  →(B1.1)
```

<u>AUTO MODE</u>

*Remark :

Following program is written for executing decision tables. Reader may look up modified algorithm presented in chapter IV. Block 'B1.' fetches the address where decision table (to be executed) is stored. Reader may recall R2 contains decision table number.

```
FLIP FLOPS  :  QA , QB
REGISTERS   :  OP(3), OTR(2)
B1.1    MAR ← (0,0,0,R2)
B1.2    MDR ← M(⊥MAR)
B1.3    PC  ← $\omega^{13}/\alpha^{15}/$ MDR
```

*Remark :

Block B2. reads a word from conditions stub. It checks for parity error. In case of error e2 is set.

Block B3. Checks the word for instruction, line marker etc. Note codes (SEN = 1 and LIN = 0). INC2 routine increments 13 bit register. It is exactly same as INCT except for some book keeping instructions.

B2.0 $DVR \leftarrow \alpha^{14}(14)$

B2.1 $MAR \leftarrow PC$

B2.2 $MDR \leftarrow M(\bot\, MAR)$

B2.3 $IR \leftarrow MDR$; $PC \leftarrow$ $\underline{INC2\ (PC)\ NO\ DELAY}$

B2.4 $e2 \leftarrow (\overline{PAR1F\ (MDR)} \vee \overline{PAR3F\ (\omega^{11}/MDR)}) \vee e2$

B3.1 $(\omega^{3}/\alpha^{4}/IR) : 1\ ,\ (<\ ,\ =\ ,\ >) \rightarrow (B5.1,\ B4.1,\ B15.1)$

*Remark :

Block B4. is <u>condition process</u> block. In step B4.1 to B4.10 conditions associated with input/output lines are processed. Address of input-output variable > 511. Properly coded address is generated in steps B4.2, B4.3 and B4.4.

B4.1 $\rightarrow (B4.2 \times IR_5 \times B4.12 \times \overline{IR_5})$

B4.2 $(\omega^{5}/\alpha^{6}/IOAR),\ (\omega^{4}/IOAR) \leftarrow (\omega^{9}/\alpha^{15}/IR)$

B4.3 $IOAR_0 \leftarrow (\overline{IR_6 \oplus IR_7 \oplus IR_8 \oplus IR_9 \oplus IR_{10}})$

B4.4 $IOAR_6 \leftarrow (\overline{IR_{11} \oplus IR_{12} \oplus IR_{13} \oplus IR_{14}})$

B4.5 $Q_A,\ Q_B \leftarrow$ SELECT1 (IOAR), SELECT2 (IOAR)

B4.6 $\rightarrow (B4.7 \times (\overline{Q_A \oplus Q_B}) + B4.8 \times (Q_A \oplus Q_B))$

B4.7 $Q_A, Q_B \leftarrow$ SELECT1 (IOAR), SELECT2 (IOAR)

B4.8 DVR $\leftarrow 2 \overset{\uparrow}{\circ}$ DVR ; $(\omega^2/\text{DVR}) \leftarrow (Q_A, Q_B)$ NO DELAY

B4.9 e1 $\leftarrow (\overline{Q_A \oplus Q_B}) \vee$ e1

B4.10 $\rightarrow$ (B2.1)

*Remark :

Block B4.12 processes conditions requiring service of IFP.

B4.12 $\rightarrow$ (SR $\times$ B4.12 + B4.13 $\times$ $\overline{\text{SR}}$)

B4.13 SR $\leftarrow$ 1 ; CAR $\leftarrow$ $(\omega^9 / \alpha^{15}/ \text{IR})$; CMR $\leftarrow$ IC1

B4.14 $\rightarrow$ (SR $\times$ B4.14 + B4.15 $\times$ $\overline{\text{SR}}$)

B4.15 DVR $\leftarrow 2 \overset{\uparrow}{\circ}$ DVR ; $\omega^2/\text{DVR} \leftarrow (\text{OUTF}, \overline{\text{OUTF}})$

B4.16 $\rightarrow$ (B2.1)

*Remark :

Condition stub is finished. We expect next location to contain address of action stub. R1 is restored.

B5.1 MAR $\leftarrow$ PC

B5.1 MDR $\leftarrow$ M($\perp$ MAR)

B5.3 R1 $\leftarrow \omega^{13}/\alpha^{15}/\text{MDR}$, NO DELAY

B5.4 PC $\leftarrow$ INC2 (PC) NO DELAY

B5.5 e2 $\leftarrow (\overline{\text{PARIF (MDR)}} \vee \text{e2})$

*Remark :

Block B6. reads a word of decision table then

Block B7. checks it for a rule Vs line marker

Block B8. performs matching operation by calling combinational function routine.

B9.1 is branch instruction.

```
B6.1    MAR ← PC
B6.2    MDR ← M(⊥ MAR)
B6.3    R3 ← $\omega^{14}/ \alpha^{15}/$ MDR ; PC ← INC2 (PC) NO DELAY
B6.4    e2 ← $\overline{\text{PAR1F (MDR)}}$ ∨ e2
B7.1      R3 : 0 , (= , ≠) → (B10.1 , B8.1)
B8.1    $\ell$ ← RULEF (R3, DVR)
B9.1    → (B10.1 × $\ell$ + B6.1 × $\bar{\ell}$)
B10.1   PC ← R1.
```

*Remark :

If ℓ = 0 then ELSE rule holds otherwise NON ELSE rule holds.

Block B11 reads a word for action stub

Block B12 makes check on opcode of instructions

Block B13 executes actions.

```
B11.1   MAR ← PC
B11.2   MDR ← M(⊥ MAR)
B11.3   IR ← MDR ; PC ← INC2 (PC) NO DELAY
B11.4   e2←($\overline{\text{PAR1F (MDR)}}$ ∨ PAR3F ($\omega^{11}$/MDR)) ∨ e2
B12.1     ($\omega^3/ \alpha^4$/IR) : 0 , (= , ≠) → (B14.1, B13.1)
```

*Remark :

Action Execution begins here.

Steps upto B13.7 handles GDE and GDT.

Note GDE is coded as $(6)_{10}$ and GDT as $(7)_{10}$.

```
B13.1    $\rightarrow OP \leftarrow \omega^3 / \alpha^4 / IR$
B13.2    $\perp OP : 6 , (< , \geq) \rightarrow (B13.8 , B13.3)$
B13.3    $\rightarrow (B13.4 \times ((\perp OP = 7) \wedge \ell \vee (\perp OP = 6) \wedge \bar{\ell}) +$
                 $B11.1 \times ((\perp OP = 7) \wedge \bar{\ell} \vee (\perp OP = 6) \wedge \ell))$
B13.4    $R2 \leftarrow \omega^{10} / \alpha^{15} / IR$
B13.5    $\rightarrow (B14.2)$
```

*Remark :

Following steps upto B13.8 operate on I/O logic Block. First bit of address portion of IR tells us if following processing is required. (SET = 4 ; CLR = 5).

```
B13.8    $\rightarrow ((IR_5 \wedge \overline{IR_6}) \times B13.9 + B13.19 \times (\overline{IR_5}) + B11.1 \times (IR_6 \wedge IR_5))$
B13.9    $(\omega^5 / \alpha^6 / IOAR), (\omega^4 / IOAR) \leftarrow (\omega^9 / \alpha^{15} / IR)$
B13.10   $IOAR_0 \leftarrow (\overline{IR_6 \oplus IR_7 \oplus IR_8 \oplus IR_9 \oplus IR_{10}})$
B13.11   $IOAR_6 \leftarrow (\overline{IR_{11} \oplus IR_{12} \oplus IR_{13} \oplus IR_{14}})$
B13.12   $OTR_0 \leftarrow (\perp OP = 4) \wedge \ell \vee (\perp OP = 5) \wedge \bar{\ell}$
B13.13   $OTR_1 \leftarrow \overline{OTR_0}$
B13.14   line outline $\leftarrow$ PULSE
B13.15   $(Q_A , Q_B) \leftarrow (SELECT1 (IOAR), SELECT2(IOAR))$
B13.16   $\rightarrow (B11.1 \times (\perp OTR = \perp(Q_A,Q_B)) + B13.17 \times (\perp OTR \neq \perp(Q_A,Q_B)))$
B13.17   $e1 \leftarrow 1$
B13.18   $\rightarrow (B11.1)$
```

*Remark :

Following steps process action requiring service of IFP.

Step B13.22 is for SET

```
B13.19    → (B.13.19 × SR + B13.20 × $\overline{SR}$)
B13.20    CAR ← $\omega^9$ / $\alpha^{15}$ / IR
B13.21    ⊥OP : 4 , (< , = , >) → (B13.26 , B13.22 , B13.24)

*Remark :  OP = SET.

B13.22    CMR ← IC5 ; OUTF, SR ← ℓ , 1
B13.23    → (B11.1)

*Remark :

          OP = CLR , Note OP > 5 have been eliminated.

B13.24    CMR ← IC5 ; OUTF, SR ← $\bar{\ell}$ , 1
B13.25      (B11.1)

*Remark :

          Comes here if OP < 4 i.e. OP = INC OR DEC ,

                         INC = 2

B13.26    ⊥OP : 2 , (0 , ≠) → (B13.30, B13.27)
B13.27    ⊥OP :  3 , (= , ≠) → (B13.33 , B13.28)
B13.28    e2 ← 1
B13.29    → (B11.1)

*Remark :  INC

B13.30    CMR ← (IC1) ∧ $\bar{\ell}$ ∨ (IC3) ∧ ℓ ; SR ← 1
B13.31    → (B11.1)

*Remark :  DEC

B13.33    CMR ← (IC1) ∧ $\bar{\ell}$ ∨ (IC4) ∧ ℓ  ; SR ← 1
B13.34    → (B11.1)
```

*Remark :

Routine to increment R2 is very similar to INC or INC2 and hence will not be given. Error statuses are noted down in memory.

```
B14.1   R2 ← INC3 (R2)
B14.2   → (B14.2 × SR + B14.3 × $\overline{SR}$)
B14.3   SR , OUTF ← 1,e2 ;  CAR, CMR ← ( $\omega^1(9)$, (IC5))
B14.4   → (B14.4 × SR + B14.5 × $\overline{SR}$)
B14.5   SR, OUTF ← 1, e1 ; (CAR , CMR) ← ($\omega^0(9)$ , (IC5))
B14.6   → (BO.2)
B15.1   e2 ← 1
B15.2   → (B14.1)
```

*Remark :

Combinational function routines now follows.

'B' denotes book-keeping statements.

```
        1. FUNCTION RULEF (R3, DVR)
'B'     2. TA, TOR ← 14ρ0, 7ρ0
        3. TA ← (R3 ∧ DVR)
'B'     4. i ← 6
        5. $TOR_i$ ← ($TA_{(2*i+1)}$ ∨ $TA_{2*i}$)
'B'     6. i ← i - 1
'B'     7. i : 0 , (≥, <) → (5,8)
        8. RULEF ← (∧/ TOR)
        9. RETURN
```

*Remark :

PAR1F 'EX-OR's all the bits. If number of bits are odd then it is 1. So if PAR1F = 1 then no error. It analyses bits 0 to 15 and hence checks parity bit 0.

```
    1.  FUNCTION PAR1F (MDR)
'B' 2.  TI, TV, TW ← 7ρ0 , 4ρ0 , 2ρ0
'B' 3.  i ← 6
    4.  $TI_i \leftarrow MDR_{2*i+2} \oplus MDR_{2*i+1}$
'B' 5.  i ← i-1
'B' 6.  i : 0, (≥ , <) → (4,7)
    7.  $TV \leftarrow (MDR_0 \oplus TI_0), (TI_1 \oplus TI_2), (TI_3 \oplus TI_4), (TI_5 \oplus TI_6)$
    8.  $TW \leftarrow (TV_0 \oplus TV_1), (TV_2 \oplus TV_3)$
    9.  $PAR1F \leftarrow TW_0 \oplus TW_1$
    10. RETURN
```

*Remark :

PAR2F in a similar manner checks for parity bit correctness of CMOUT. PAR2F = 1 if no error.

```
    1.  FUNCTION PAR2F (CMOUT)
'B' 2.  TI ← 2ρ0
    3.  $(TI_0, TI_1) \leftarrow (CMOUT_0 \oplus CMOUT_1), (CMOUT_2 \oplus CMOUT_3)$
    4.  $PAR2F \leftarrow (TI_0 \oplus TI_1) \oplus CMOUT_4$
    5.  RETURN
```

*Remark :

PAR3F checks parity bit 15 of a 16 bit register.

Output = 1 if no error.

```
      1.  FUNCTION PAR3F ($\omega^{11}/MDR$)
'B'   2.  TV, TW ← 5ρ0, 3ρ0
'B'   3.  i ← 4
      4.  $TV_i \leftarrow (\omega^{11}/MDR)_{2*i+1} \oplus (\omega^{11}/MDR)_{2*i+2}$
'B'   5.  i ← i-1
'B'   6.  i : 0 , ($\geq$, <) → (4,7)
      7.  $TW \leftarrow ((\omega^{11}/MDR)_0 \oplus TV_0), (TV_1 \oplus TV_2), (TV_3 \oplus TV_4)$
      8.  $PAR3F \leftarrow (TW_0 \oplus TW_1) \oplus TW_2$
      9.  RETURN
```

*Remark :

Routine to increment PC now follows :

Note : $\wedge/\omega^0/PC = \wedge/\bar{\alpha}^0/PC = 1$

```
      1.  CL SUBROUTINE INC2 (PC)
'B'   2.  CTNEW ← 13ρ0
'B'   3.  i ← 12
      4.  $CTNEW_i \leftarrow PC_i + (\wedge/(\omega^{12-i}(13)/PC))$
'B'   5.  i ← i-1
'B'   6.  i : 0 , ($\geq$, <) → (4,7)
      7.  INC2 (PC) ← CTNEW
      8.  RETURN.
```

<u>Program Entry Mode</u>

All statements in this program begin with P. All registers and flip-flops declared earlier under the program are also available.

*Remark :

In step PO.0 PLC waits for command from KBP.

In step PO.1 PLC becomes busy. So rdycon is set to zero. PARITY is checked and if error, branching to PE1.0 takes place. PO.3 onwards decodes the command.

FLIP FLOPS : f2, cf1, cf2

```
PO.0    →  (PO.0 × $\overline{\text{Cavoutf}}$ + PO.1 × cavoutf)
PO.1    DATAR ← IOR ; rdycon, cavoutf ← (0,0)
PO.2    f2 ← PAR1F (DATAR) ∧ PAR2F (CMOUT) ∧ PAR3F ($\omega^{11}$/DATAR)
PO.3    →  (PO.4 × f2 + Pe1.0 × $\overline{\text{f2}}$)
```

*Remark :

Commands are decoded below. The commands C1, C2, C3, C4, C5 return through a common subprogram (beginning at PC1.0) toBO.2 which sees if stop is pressed or not ? Then transfers the control to appropriate place.

```
PO.4    CMOUT : C1 ,(= , ≠)  →  (P1.1, PO.5)
PO.5    CMOUT : C2 ,(= , ≠)  →  (P2.1, PO.6)
PO.6    CMOUT : C3 ,(= , ≠)  →  (P3.1, PO.7)
PO.7    CMOUT : C4 ,(= , ≠)  →  (P4.1, PO.8)
PO.8    CMOUT : C5 ,(= , ≠)  →  (P5.1, PO.9)
PO.9    CMOUT : C6 ,(= , ≠)  →  (P6.1, PO.10)
PO.10   CMOUT : C7 ,(= , ≠)  →  (DP7.1,PO.11)
PO.11   CMOUT : C9 ,(= , ≠)  →  (P9.1, PE1.0)
```

*Remark : Command C1 : Load data into PC.

```
P1.1    PC ← $\omega^{13}$ / $\alpha^{15}$ / DATAR
P1.2    →  (PC1.0).
```

*Remark :

Treat data as DT number. cf1, cf2 are used to remember past history. cf1, cf2 are used to branch to different portions inside 'P3.' block. Note action routine address is set by PLC itself. FUNCTIONS PAR1F or PAR2F, or PAR3F can be used to generate parity bits if we set parity bits to 1 before calling them. If parity bit set is correct then you get 1 as output. (i.e. correct parity bit). If parity bit is wrong, you get 0 output which is again desired parity bit.

P2.1 MAR $\leftarrow (\omega^{13} / \alpha^{15}$ / DATAR) ; cf1, cf2 $\leftarrow$ (0,1)

P2.2 MDR $\leftarrow$ 1, 0, PC, PAR3F (((ω^{10}/PC),1))

P2.3 $\text{MDR}_0 \leftarrow$ PAR1F (MDR)

P2.4 M($\perp$ MAR) $\leftarrow$ MDR

P2.5 $\rightarrow$ (PC1.0)

*Remark : Command C3 : A word to be loaded into memory.

P3.1 MDR $\leftarrow$ DATAR ; MAR $\leftarrow$ PC ; PC $\leftarrow$ INC2 (PC)

P3.2 M($\perp$ MAR) $\leftarrow$ MDR

*Remark :

cf1, cf2 = 0,1 and MDR = 0 ==> next location's address to be saved for storing action routine address which will be known later.

P3.3 $\rightarrow$ (P3.4 $\times$ ($\overline{\text{cf1}}$ $\wedge$ cf2 $\wedge$ ($\perp(\omega^{14}/ \alpha^{15}$/MDR) = 0))

+ P3.6 $\times$ (cf1 $\wedge$ cf2 $\wedge$ ($\perp(\omega^{14}/ \alpha^{15}$/MDR) = 0))

+ PC1.0$\times$($\overline{\text{cf2}}$ $\vee$ ($\perp(\omega^{14}/ \alpha^{15}$/ MDR) $\neq$ 0)))

```
P3.4    R1 ← PC ; PC ← INC2 (PC) ; cf1 ← 1
P3.5     → (PC1.0)
```

*Remark :

cf1, cf2 = 11 implies rules are being entered. If in addition MDR = 0 then rules are finished. Actions are expected from next time.

```
P3.6    MDR ← (1,0,PC,1) ; PC ← R1 ; cf2 ← 0
P3.7    MAR ← PC ; PC ← $\omega^{13}/ \alpha^{15}/$ MDR
P3.8    $MDR_0$ ← PAR1F (MDR) ; $MDR_{15}$ ← PAR3F ($\omega^{11}$/MDR)
P3.9    M( ⊥ MAR) ← MDR
P3.10    → (PC1.0)
```

*Remark :

Command C4 : Data to be loaded in CAR.

```
P4.1     → (P4.1 × SR + P4.2 × $\overline{SR}$)
P4.2    CAR ← $\omega^9/ \alpha^{15}/$ DATAR
P4.3     → (PC1.0)
```

*Remark :

Command C5 : Load DATAR in CAR and ask IFP to load it into its memory.

```
P5.1     → (P5.1 × SR + $\overline{SR}$ × P5.2)
P5.2    CDR ← DATAR ; CMR ← (IC6) ; SR ← 1
P5.3     → (PC1.0)
```

*Remark :

Command C9 : Supply PC content.

```
P9.1    $\omega^{13}/\ \alpha^{15}/$ DATAR ← PC ; CMIN ← CR3
P9.2    → (PC1.0)
```

*Remark :

Common return program follows :

```
PC1.0   CMIN ← CR2
PC1.1   → (rdyp × PC1.2 + $\overline{\text{rdyp}}$ × PC1.1)
PC1.2   line cavin ← PULSE ; rdycon ← 1
PC1.3   → (B0.2)
```

*Remark :

Display of DECISION TABLE

We have used a MACRO 'OUT' only to avoid repeating very similar statements. This makes it easier to understand the program. First three steps fetch decision table address.

```
P6.1    MAR ← ($\omega^{13}/\ \alpha^{15}/$ DATAR) ; cf1, cf2 ← (1,1)
P6.2    MDR ← M($\perp$ MAR)
P6.3    PC ← ($\omega^{13}/\ \alpha^{15}/$ MDR)
```

*Remark :

For meaning and listing of 'OUT' see at the end of this program for display of DT . cf1, cf2 = 11 means we are still displaying condition stub. cf1, cf2 = 1,0 means condition stub is finished. Action routine address to be outputted. After displaying any word MC expects a 'proceed' command to ensure that either address is loaded into DL or user has observed the one word of information.

For each word in decision table (whether the word be condition, or rule, or action routine address or action) its location in MC's memory has to be transmitted to KBP. This function is served by steps P6.4 to P6.7.

```
P6.4    OUT (PC,3)
P6.5    → (P6.5 × $\overline{cavoutf}$ + P6.6 × cavoutf)
P6.6    cavoutf ← 0 ; rdycon ← 0.
P6.7    CMOUT : C8 , (= , ≠) → (P6.10, PE1.0)
```

*Remark :

Next block reads a word and checks it for zero. cf1, cf2 = 01 and MDR = 0 implies action stub is finished. Return-program starts at P6.30.

```
P6.10   MAR ← PC
P6.11   MDR ← M(⊥ MAR) ;  PC ← INC2 (PC)
P6.12   ($\omega^{14}/ \alpha^{15}/$ MDR) : 0 , (= , ≠) → (P6.14, P6.13)
P6.13   →(cf2 × P6.17 + (cf1 ∧ $\overline{cf2}$) × P6.21 + ($\overline{cf1}$ ∧ $\overline{cf2}$) × P6.25)
P6.14   cf2 ← $\overline{cf2}$
P6.15   → (($\overline{cf1}$ ∧ $\overline{cf2}$) × P6.30 + (cf1 ∨ cf2) × P6.4)
```

*Remark :

Following subportion is meant to output <u>action</u> or <u>condition</u> stub of DT to KBP.

```
P6.17   OUT (MDR,4)
P6.18   → (cavoutf × PC6.19 + $\overline{cavoutf}$ × PC6.18)
P6.19   cavoutf, rdycon ← (0,∪)
P6.20   CMOUT : C8 , (=, ≠) → (P6.4 , PE 1.0)
```

*Remark :

It out-puts action stub address. cf1, cf2 = (0,0) means rules are being outputted. Hence cf1, cf2 made (0,0)

```
P6.21    OUT (MDR,6)
P6.22    cf1, cf2 ← (0,0)
P6.23    → (P6.18)
```

*Remark :

Outputs <u>RULES</u>

```
P6.25    OUT (MDR,5)
P6.26    → (P6.18)
```

*Remark :

This program sends a word containing 0's to KBP. This tells the operator 'end of DT'.

```
P6.30    OUT (MDR, 6)
P6.31    → (BO.2)
```

*Remark :

MACRO : OUT (A,i)

This macro loads register DATAR by content of A. Command generated for KBP is CRi - Step 3 pulses the cavin line to inform KBP about command and data. The assignment statement 'DATAR ← A' has to be taken with caution. If length of register A is smaller then appropriate bits in DATAR should be made zero.

MACRO OUT (A,i)

```
1.    DATAR ← A ;  CMIN ← CRi
2.    → (rdyp × 3 + $\overline{rdyp}$ × 2)
```

```
3.      line cavin ← PULSE ; rdycon ← 1
4.      RETURN.
```

*Remark :

```
           ERROR ROUTINE:
PE1.0      CMIN  ← CR1
PE1.1       → (PC1.1)
```

6.8 TAIL PIECE OF AHPL PROGRAM FOR PIC

We mentioned in sections 6.6 and 6.7 that MC and KBP program branches to step W4. This finish off program is now presented. At step W4 we converge the above mentioned two process. Convergence step waits till both process arrive at that step. The program follows:

```
W4     CONVERGE (BO.2, KO.3)
W5     stopf ← 0
W6      → (W1)
```

CHAPTER VII
LSI IMPLEMENTATION OF PLC

The advent of the microprocessor is the beginning of a new computer revolution in our technological society. The multitude of companies (in U.S.A.) announcing microprocessor developments, the rapidly increasing discussion of them in magazine articles and conferences, and the increasing number of products using them indicates the revolution is well underway.

Microprocessors are Large Scale Integration (LSI) devices and traditionally refer to central processing unit of a computer. Therefore to construct any computer, input/output units and memory has to be added to the microprocessor chip. We can refer to such a computer as a "micro-computer".

Using micro-processors to construct systems, system manufacturing costs are reduced due to lower cost of packaging (fewer connectors, printed circuit boards, cables, and so forth), smaller power supply, lower assembly costs, more modest cooling requirements, and greater standarization of hardware. Product variations may be achieved by program change rather than by logic design modifications.

7.1 SELECTION OF A MICROPROCESSOR

Specs don't tell all (17) -

Unlike the less complicated IC's microprocessors can not be completely characterized on a simple data sheet. Moreover different vendors use different parameters to measure a processor's cupabilities. This makes a selection of the best one a difficult task. However we shall arrive at some criteria to select a microprocessor.

Input/Output Capability -

This area needs very careful consideration. I/O may impose a serve limitation on the system requiring heavy input/output activity. The input-output instructions available, the number of ports it can access, the time taken to access a port, time taken to generate an address, all have to be considered.

Interface hardware required to connect input/output devices have to be considered. Are address and data bus out puts TTL compatible ? Can they drive TTL loads ? How much additional hardware is required ? Is any interface chip available ? And similar other questions are to be answered.

Availability of General Purpose Registers -

The quality, not the quantity is a very important factor which influences the execution time of a program. Questions to be answered are :

Can all registers be incremented/decremented or only accumulator is equipped to do so ?

Can every register be tested for zero or only accumulator is equipped to do so ?

etc.

If quantity and quality are both there then memory references and inter-register data movements can be minimized leading to considerable improvement in execution time. Careful analysis of instruction sheet may give us the above information.

<u>Interrupt Capability (17)</u>

Majority of manufacturers claim some type of interrupt handling ability for their chips. The service performed by chips on receiving a interrupt may vary a lot over different microprocessor chips. For some of them there is so much burden put on software that it discourages a designer from attempting to handle even a single interrupt besides time taken to handle the interrupt. But newer microprocessor chips can accomodate single, multilevel and vectored interrupts and they save essential registers automatically into a push down stack and branch to a specified location in the memory.

<u>Analyse the Instruction Set Available</u> -

A variety of addressing modes (e.g. immediate, indexed, indirect, indexed with indirection) are possible. Availability of these influence the length of program and execution time. Indexing is very convenient to fetch table of data, or to fetch a location which is not known a priori but whose address depends upon some calculation or conditions.

Another thing to be checked is about the availability of frequently used operations. If an operation which is used very frequently in an application, is missing from the instruction sheet of the microprocessor then that microprocessor is likely to perform poorly.

Microprogrammability -

Instruction set of a microprogrammable microprocessor can be altered. Missing operations may be supplied and performance improved. In essence microprogramming can be used to customize the design to one's requirements.

We shall now look up some of the microprocessors and see where they stand.

7.2 ANALYSIS OF MICROPROCESSORS

Intel's MC S-4 (18)

It is a 4 bit micro-computer chip set manufactured by Intel corporations. It employs P-channel Silicon Gate MOS technology and has the distinction of being the first microprocessor to appear in the market. The MCS - 4 uses a 10.8 micro-second instruction cycle. This micro-computer set consists of the following 4 chips, each packaged in a 16 pin DIP package :

(1) A Central Processor Unit Chip - CPU - 4004

(2) A Read Only Memory Chip - ROM - 4001

(3) A Ramdom Access Memory Chip - RAM - 4002

(4) A Shift Register Chip - SR - 4003.

One CPU controls upto 16 ROM's (16 × 256 × 8 bits), 16 RAM's (16 × 80 × 4 bits) without requiring any interface circuit. The I/O function, although different from ROM and RAM functions, is physically located in the ROM and RAM chips. Each 4001 has a 4 bit input-output port. Each 4002 has a 4 bit output port and information can be routed only out of it. All communication between CPU and I/O devices must take place through these I/O ports provided with RAMs, ROMs, or SRs. The MCS-4 is not suitable for our application because of the following reasons :

Sever I/O Limitation

Ports which only output the information, are not useful for us. Total number of remaining ports are 16. Even if we use additional interface circuitry we can extend total number of ROM ports (capable of inputting as well as out-putting) to only 24. We require a total number of input-output ports to be around 500. Shift register can not be used to increase the number of lines as it slows down input output tremendously. (Shift registers are provided to connect slow devices like key boards, printers etc.) Therefore MC S-4 can not be used.

It is not easy to access a port. One has to first select that particular RAM or ROM chip and may require ∿20 μsecs. Then one has to execute input - output instructions. It increases the access time for a line and hence MCS-4 is not acceptable to us.

Accumulator is the only general purpose register.

4 bit data word will mean lots of software problems to store and execute decision tables.

Hence MCS-4 is not suitable for our application.

<u>8080 Microcomputer</u>

The 8080 is a 8-bit central processing unit (CPU) for use in <u>general purpose</u> computer systems (19). It is fabricated on a single chip using n-channel silicon gate MOS technology.

Instructions in the 8080 contain one to three bytes. Each instruction requires from one to five machine or memory cycles for fetching and execution. Each machine cycle requires from 1.5 μsec to 2.5 μsec (2MHZ maximum clock frequency is assumed). Instruction time requirements range from a minimum of 2 micro-secs for non-memory referencing instructions like register and accumulator arithmetic instructions, upto a maximum of 9 micro-secs for the complex instructions.

Intel 8080 contains six 8-bit data registers, an 8-bit accumulator, four testable flag-bits, and an 8-bit parallel binary arithmetic unit. The four testable flay-flip-flops are carry, zero, sign and parity. They are set under following conditions :

carry	- Overflow, underflow
zero	- Result is zero
sign	- Most significant bit of result is "1"
parity	- Parity of result is even.

Analysis of Relevant Instructions -

Meaning of the symbols used to describe instructions are presented in Figure 7.1 and relevant instructions are presented in Figure 7.2.

Symbols	Meaning
$<B_2>$	Second Byte of the instruction
$<B_3>$	Third Byte of the instruction
r	One of the 6 scratch pad register references : A, B, C, D, E, H, L
M	Memory location indicated by the contents of registers H and L
( )	Contents of a location or a register
A_m	Bit m of the A- register (Least significant bit of the register is 0th bit)
	Logical product
PC	Program Counter
$\leftarrow$	"Is replaced by"
XXX	A "don't care"
SSS	Source register for data
DDD	Destination register for data

Register # (SSS or DDD)	Register Name
000	B
001	C
010	D
011	E
100	H
101	L
110	Memory
111	A or Accumulator

Figure 7.1

Mnemonic	Bytes	Time in micro-secs	Description
MOV r_1,r_2	1	2.5	$(r_1) \leftarrow (r_2)$
MVI r $<B_2>$	2	3.5	$(r) \leftarrow <B_2>$
MOV r,M	1	3.5	$(r) \leftarrow (M)$ (M) is addressed by the contents of registers H and L
INR r	1	2.5	$(r) \leftarrow (r) + 1$ All flags except carry are affected by the result
DCR r	1	2.5	$(r) \leftarrow (r) - 1$ All flags except carry are affected by the result
ANA M	1	3.5	$(A) \leftarrow (A) \wedge (M)$ Logical AND (M) is addressed by the contents of registers H and L
SUI $<B_2>$	2	3.5	$(A) \leftarrow (A) - <B_2>$ SUBTRACT IMMEDIATE
JC $<B_2>$ $<B_3>$	3	5	If (carry) = 1 $(PC) \leftarrow <B_3> <B_2>$ Otherwise (PC) = (PC) + 3
JNZ $<B_2>$ $<B_3>$	3	5	If (zero) = 0 $(PC) \leftarrow <B_3> <B_2>$ Otherwise (PC) = (PC) + 3
JNC $<B_2>$ $<B_3>$	3	5	If (carry) = 0 $(PC) \leftarrow <B_3> <B_2>$ Otherwise (PC) = (PC) + 3
IN $<B_2>$	2	5	$(A) \leftarrow$ (Input data) $<B_2>$ denotes the I/O device number.
OUT $<B_2>$	2	5	(Output data) $\leftarrow (A)$ $<B_2>$ denotes the I/O device number.
RLC	1	2	$A_{m+1} \leftarrow A_m$, $A_0 \leftarrow A_7$, $(carry) \leftarrow A_7$. Rotate the content of registers A left 1 bit.

Table of 8080 Instructions

Figure 7.2

We are now ready to analyse how Intel 8080 fares in our application. The modified algorithm presented in Chapter IV requires a special kind of bit operation which is not available in the microprocessor chip and so this algorithm is not suitable for implementation on the chip. Instead the algorithm published by Muthukrishnan and Rajaraman (11) is appropriate. It requires 'AND' operation which is available on the 8080.

Convenient scheme to store a decision table would be to store conditions' addresses in the first few locations, followed by condition entries row-wise and thereafter action stub of the decision table can be store. Each action has to be coded appropriately. There has to be an executive program whose function is to generate the data-vector, find the rule satisfied, take care of actions and then take next decision table for execution. Number of pointers and data-elements will be required. Examples of pointers are 'beginning address of a decision table', and 'address of next location to be accessed'. Examples of data-elements are number of conditions in a decision table, number of actions in the action stub etc. To simplify matters let us put the restriction on the decision table size. Let maximum number of rules be equal to or less than 8 and maximum number of conditions be equal to or less than 4 (Note : register length is 8 bits and result of each condition - test requires two bits. Memory word length is also 8 bits). We shall not write a complete executive program but consider a portion of it which finds the rule satisfied. We shall assume that registers H, L, B, C, D contain the following quantities before we enter the program (that we are going to write).

H,L	contains	the address of first location of condition entries.
B	contains	condition count.
C	contains	data vector
D	used	for result vector. Initially contains a vector of all 1's.

The program is presented below :

Location	Instruction	Comments	Time in micro-secs
1	MOV A, C	(A) ← (C)	2.5
2	RLC	Rotate left A (data vector) one bit	2
3	JNC 12	If carry bit = 0 then jump to location 12 (i.e. the corresponding condition was false).	5
4	RLC		2
5	MOV C,A	Shifted data vector is stored	2.5
6	MOV A,D		2.5
7	ANA M	Result vector 'AND'ed with a row of condition entries	3.5
8	MOV D,A		2.5
9	INR L	It is ignored that L can over-flow (L) ← (L) + 1	2.5
10	INR L		2.5

Location	Instruction	Comments	Time in micro-secs
11	JMP 19	Jump to 19 .	5.0
12	RLC	Comes here if the corresponding condition was false.	2.0
13	MOV C,A		2.5
14	MOV A,D	Result vector Loaded in A.	2.5
15	INR L		2.5
16	ANA M	Result vector 'AND' ed with a row of condition entries.	3.5
17	INR L		2.5
18	MOV D,A	Result vector restored.	2.5
19	DCR B	Decrease condition count by 1.	2.5
20	JNZ 1	Jump to 1 if B not equal to zero.	5.0
21	MVI B,B	B initialized to number of rules = 8.	3.5
22	MOV A,D	Result vector moved to A.	2.5
23	RLC		2
24	JC XY	If carry = 1 i.e. a rule holes jump to location XY.	5.0
25	DCR B		2.5
26	JNZ 23	Test B for zero. If not equal to zero; to to 23.	5.0
27	....	Comes here if ELSE rule holds.	
⋮			
XY	....	Comes here if a rule other than ELSE holds.	

In case a condition is true steps 1 through 20 once, take 40 micro-secs. In case a condition is false steps 1 through 20 once take 35 micro-secs. For a 4 condition decision table it will take on average 150 micro-secs (Arbitrarily assumed that true and false conditions occur with equal probabilities. It does not affect the result much). If we assume that on average we come out of the program (from instruction in location 24) after two rules then total time to find a rule on average

= 150 + 43.5 = 193.5 micro-secs.

Therefore it takes 193.5 micro-secs to find a rule which holds. In case of our PLC it takes only 4 clock pulses to test if a rule is satisfied or not satisfied. If we assume 2 MHZ clock and 2 rules on the average we get the total time = 4.0 micro-secs. Hence the performance of Intel 8080 (193.5 compared to 4) is bad indeed. The reasons for such a degradation are :

Executive program is required. In case of hardwired processor like our PLC the executive program could be eliminated saving in fetching time of instructions, and eliminating the time taken by jump or branch instructions.

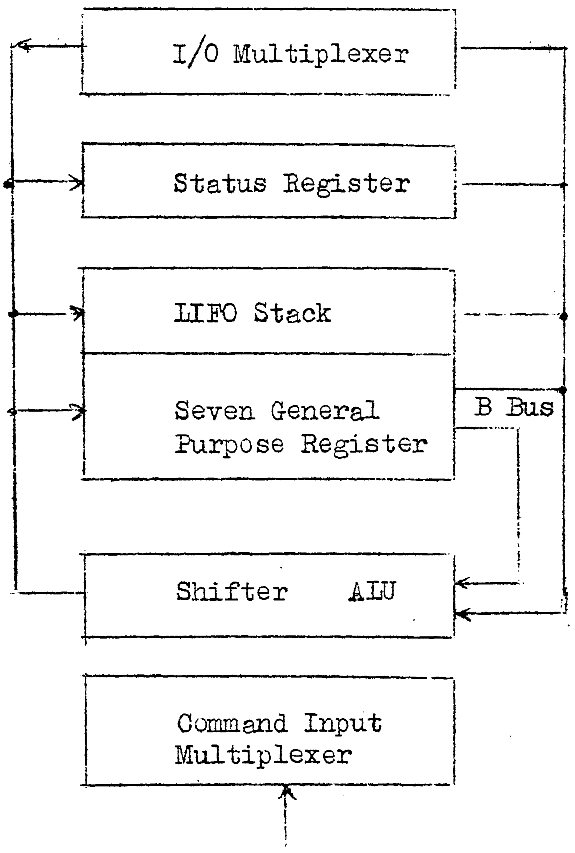
Only one general purpose register (Accumulator). This results in unnecessary movements of data.

If we allow more than 4 conditions in a decision table then the performance will be degraded still further. Therefore we conclude that the microprocessor, in the portion of executive program that we analysed, is order of magnitude slower than our PLC. But how does it fares in other portions. We shall examine this now. There are two approaches to process rest of decision tables. One is to store for each conditions it address and for each action its address and code indicating type of the action. The other is to compile the condition and action stubs before entering a decision table and store condition and action stubs as condition and action routines respectively. In the first approach one has to perform software decoding of action code-field which will involve test and shift of a register. The other overheads required to fetch status of a input-output line, or any other variable or function are same as in the second approach. Therefore this microprocessor implemented PLC is order of magnitude slower than our PLC. If we follow the second approach which requires a separate computer facility to be available then also in case of condition routine unnecessary data movements, shifts and ORing which are not performed in our PLC, are performed here. In case of action routine also it can not fare better because every data to be communicated to memory or I/O devices, has to be placed in the accumulator before we execute I/O instructions. Hence we conclude that Intel 8080 microprocessor chip can not serve our purpose.

National's IMP (20)

IMP bhips are manufactured by National Semiconductor Corp. The IMP velongs to multichip CPU class of microprocessor.

The IMP data processing chip (the Register and Arithmetic-Logic Unit or RALU) provides a four-bit slice of each of the following functions: seven data registers, a four-function ALU with complementing input, a data shifter, a status register, a 16-word last-in-first-out (LIFO) stack, and input-output register (See Figure 7.3).



Block diagram of RALU
Figure 7.3

Upto eight of these chips may be used to implement processors whose word lengths are multiples of four bits. All RALU's operate in parallel under the control of the second chip type (the control and Read Only Memory or CROM) to form a synchronous parallel processor. The operational characteristics of the processor are primarily determined by a microprogram stored in the CROM. This microprogram specifies the assignment of seven RALU registers, the instruction set implemented by the processor, the timing and function of control signals in the processor interface (e.g. data bus control, interrupt requests), and also allows for different word lengths which can be implemented with RALU (21).

IMP is a monolithic MOS/LSI circuit utilizing standard P channel, enhancement mode, silicon gate technology. Signals which are intended for interface with the CROM are MOS level, while those which are intended for interface with the rest of the processor system are TTL levels.

We shall analyse 16-bit microprocessor systen provided by National Semiconductor Corporation. In the absence of micro-programming manual (Pub. No. 4200 001) it is not possible to design our own microprogrammed processor system. However we shall estimate if we would have microprogrammed the microprocessor for our application.

In the booklet (Pub. No. 4200005B) total time of execution of a instruction is given to be= (1..00N + 0.00W + 0.25R)micro-secs for 1 micro-second microprogram cycle as a typical value. Here N isthe number of microprogram cycles, W is the number of main memory write cycles, R is the number of memory read cycles. The execution time mentioned above includes instruction fetch time (2.25 micro-secs) from system memory. We shall now write the sample program to find the rule of a decision table with the constraints mentioned earlier.

The seven data registers are used in the following fashion.

Program Counter (PC)

Memory Data Register (MDR)

Memory Address Register (MAR)

4 General-purpose working registers (AC0, AC1, AC2, and AC3) - accessible to the macroprogrammer.

Let us assume that registers AC0, AC1, AC2, AC3 have been assigned the following purpose.

AC0	Result vector (Initially all 1's)
AC1	Contains the number of conditions
AC2	Contains data vector.
AC3	Initially contains the address of first row of condition entries. It is used as a Index Register.

Assume that location "Zero" contains 0. The notations used to describe instructions are presented below:

ACr	:	Denotes specific working register (AC0, AC1, AC2, AC3) where 'r' is number of accumulator referenced in instruction.
EA	:	Denotes effective address specified by instruction directly, or by indexing.
xr	:	When not zero, this value designates number of register to be used in indexed mode.

The program is presented below:

Location	Mnemonic	r	xr	disp	Comments	R	W	N
1	SKG	2	0	zero	If(ACr) > (EA) (PC) ← (PC) + 1	2	-	6,7
2	JMP		0	7	Jumps if condition (1,0) was true	1	-	3
3	AISZ	3		1	Comes here if condition was fulse; AC3 ← AC3+1	1	-	4,5
4	AND	0	3	0	Result vector 'AND' ed with a row of condition entries	2	-	5
5	AISZ	3		1		1	-	4,5
6	JMP		0	9		1	-	3
7	AND	0	3	0	comes here if condition was true			
8	AISZ	3		2	AC3 ← AC3+2			
9	SHL	2		2	shift left AC2 (data vector) by 2 bits	1	-	10

Location	Mnemonic	r	xr	disp	comments	R	W	N
10	AISZ	1		-1	AC1← AC1-1, skip if AC1 = 0	1	-	4,5
11	JMP		0	1		1	-	3
12	SHR	0		1		1	-	7
13	BOC				To test if AC0 $\geq$ 0, and accordingly branch takes place. We come to know if ELSE rule or other rules are satisfied	1	-	4,5

Note: In case of instructions SKG, AISZ, BOC second value mentioned in column 'N' applied if skip or jump occurs.

In case a condition is true steps 1 through 11 require 10 read cycles (R) and 40 total cycles (N). Thus the time taken = 42.5 micro-secs. In case a condition is false steps 1 through 11 require 9 read cycles and 35 total cycles. Thus the time taken = 37.25 micro-secs. We may take the arithmetic mean of these two numbers to arrive at a single number. For four conditions (under the same assumptions as made in case of Intel 8080 chip) the total time to find a rule equals 171.5 micro-secs. Hence we see that National's IMP-16 performs marginally better Intel 8080.

But we have still not explored the possibility of microprogramming the IMP ourselves. We in this case may store complete sequence of operations for finding a rule in the CROM itself. It is clear that we shall be able to save at least the time for fetching instructions from the system memory. It takes 2.25 micro-secs for fetch-routine (residing in the CROM) to access an instruction from system memory (20). Hence the time to find a rule will be less than 100 micro-secs (to be precise 99.5 micro-secs). This figure of 100 micro-secs is still quite high as compared to 4 micro-secs for our PLC.

We can microprogram the IMP for handling the condition and action stubs of the decision tables also. As explained earlier in this chapter, better approach would be to compile the action and condition stubs. The performance will be better than that of Intel 8080 but for the same reasons it can not be better than our PLC. (Note that one can not perform operationslike shift a register two bits left and load two status bits in right most two bits of the register, all in one clock pulse).

IMP can address upto 256 input and 256 output ports and so the requirement of 256 input and 256 output lines is satisfied.

We conclude that IMP though better than Intel 8080, does not serve our purpose.

7.3 INTEGRATION OF THE PLC DESIGN

In the last section we found that it is not possible design a programmable logic controller which uses decision tables as the input language. We therefore analyse the possibility of integrating our design. There are two major factors to be considered while arriving at a LSI circuit. They are:

1. Total number of leads to be brought out, and
2. packaging density of the components in the chip.

With the present Technology, there is no problem in bring out 40 leads. Rock well International are having 42 pins in their PPS-4 microprocessor chip and Intel 40 pins in their 8080 microprocessor chip. Total number of gates in such a chip generally are 8000. (MOS/LSI technology is assumed) Technology is making progress and so in future more gates and more pins per chip would become feasible.

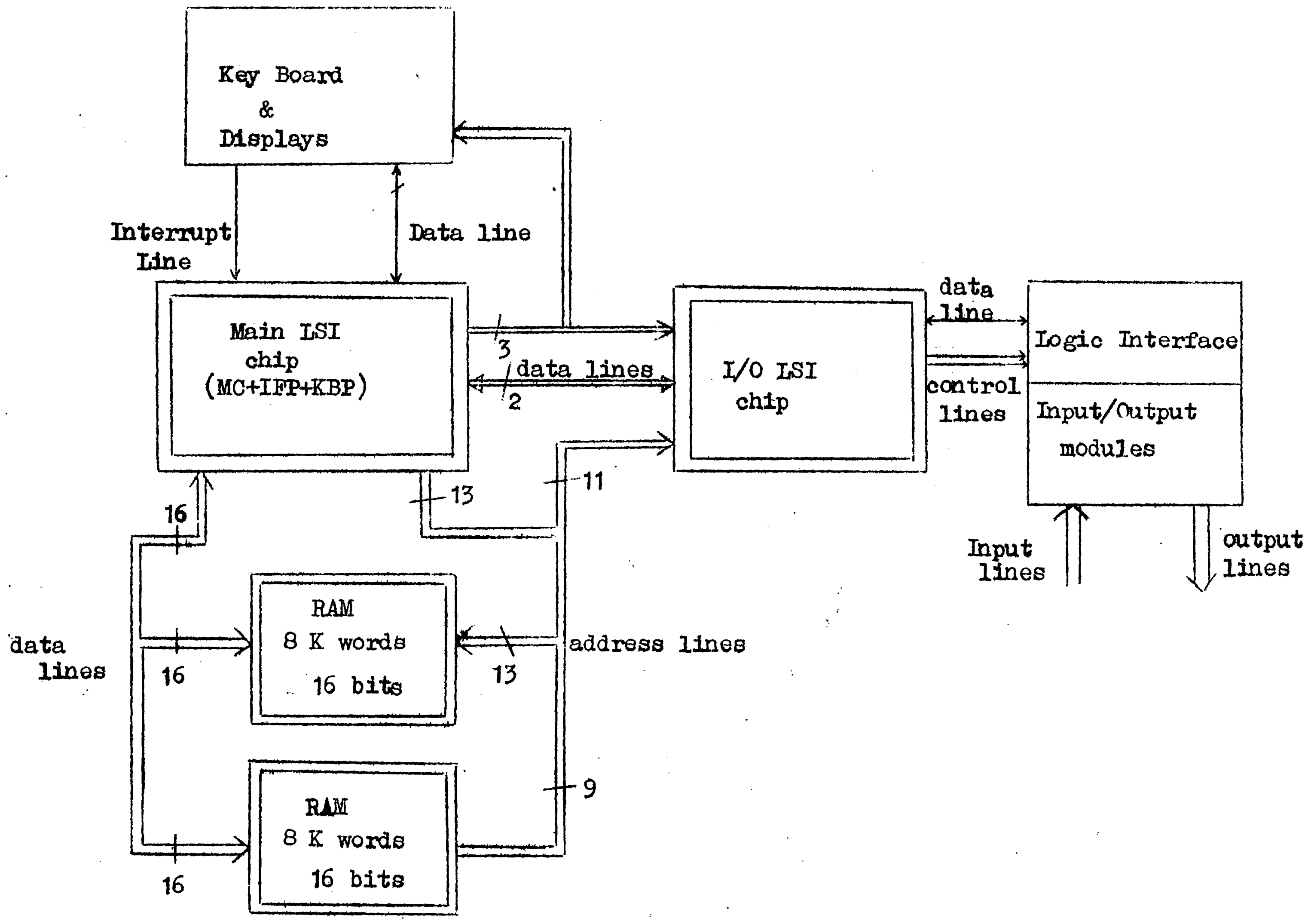
If we examine our PLC design it turns out that the complete digital circuit requires more than 10,000 gates. Hence we shall break the PLC into two LSI chips. The block diagram of this suggested PLC is shown in Figure 7.4

Input/Output Modules - These modules as explained in earlier chapters change high AC or DC input voltages to low level DC logic signals and low level DC logic signals to high voltages to control output loads.

Logic Interface Module -

It consists mainly of shift registers (total bits at the maximum 512). It is connected to I/o LSI chip by 3 lines (1 bidrectional data line and two control lines). The control lines serve the following three purposes:

1. Transfer status of Input lines in parallel to the input shift register.



Integrated PLC Block Diagram
FIGURE 7.4

2. Shift the register into data line so that bits are transferred to I/O LSI chip.
3. Accept the bits from I/O LSI chip and store serially into output shift register.

<u>I/O LSI chip</u>.

It is connected to Main LSI chip by 11 bit address line, 3 bit control line, and 2 bit data line. Under the control of 3 bit control lines, it perfoms the following functions:

1. Supplies the status of the addressed input or output line.
2. Sets the addressed output bit (line).
3. Asks the Logic Interface modules to perform the three functions mentioned under the heading 'Logic Interface Modules' .

Most of the logic is the same as designed for PLC in the last chapter. Circuit to decode and encode the control lines have to be added. A 256 bits shift register has to be added to the earlier design. Examining this design including the modification of I/O logic block presented earlier we find that the total number of gates required are

$$= \sim 5450.$$

This is well below 8000 mark and can be easily fabricated.

Key Board Display -

It has shift registers to store the status of Keys, switches and displays. All the KBP logic has been transferred to Main LSI chip. The 3 bit control line which controls I/O LSI chip, also controls this unit. 1 bit biderectional data line is provided to transfer the contents of the registers in and out of it. A single interrupt line is also provided to indicate the Main chip that the Key Board needs to service.

<u>Main LSI Chip</u> -

It represents all the logic designed for MC, IFP and KBP. 16 line data bus and 13-line address bus connects it to an 8K (16 bits/word) system RAM and a 512 words (16 bits/word) RAM (for IFP). This 13-line address-bus is also used for addressing the input/output bits in I/O LSI chip. Some changes have to be incorporated made to resolve the conflict between IFP and MC when they try to access the data and address buses simultaneously. On examining the design, we find that the total number of gates required for IFP, MC, KBP are respectively 1000, 2450 and 2280. Hence total number of gates are = 5730 which is well below 8000 mark.

Let us count the number of pins required for the Main chip.

Total number of lines connected to Main chip and shown in the Figure are	= 36
For power supply	= 3
For two phase clock	= 2
For Power-ON push button	= 1
For panel switches	= 2
Total number of pins	= 44

44 is marginally larger than 42 and is not a restriction in the view that technology ismaking progress and therefore the main LSI chip can also be manufactured. Pin requirement for I/O logic chip is considerably lower than 40.

Therefore we conclude that both LSI chips satisfy both requirements and hence the corresponding circuits can be fabricated as MOS/LSI circuits.

CHAPTER VIII

CONCLUSION

The objective of this thesis was to design a programmable logic controller which uses decision tables as a description language for control logic. Our work has been directed towards analysing some relay systems, representing them as a set of decision tables, looking at the merits and demerits of this representation, and designing the hardware for programmable logic controller.

We have developed a new model for relay control systems (cf : Chapter II). This model separates out relays, indicators, and various contacts from their interconnections which determine the operating sequence of the controller. Wiring has been a major problem for control engineers as it consumes considerable time. It was thus felt that it should be possible to develop a methodology to arrive at an interconnection network with the help of a computer and therefore reduce the design time for wiring and eliminate mistakes in the design. One may go one step further and design a machine which uses some sort of punched tape obtained as an output from the computer, and produces the wired network. This will reduce the set up time for the system and make it more reliable by eliminating human errors.

This model helps one in understanding a problem and evolving decision tables for relay control applications.

Decision tables help one to concentrate more on the decision process. It is a convenient method to represent control logic. The conditions in the decision tables may be rigorously examined and oversights minimized.

The restrictions put on decision tables have made them more useful. Benefits derived from such restrictions are : significant improvement in their speed of execution and overall saving in storage requirements.

We have designed programmable logic controller and the design has been documented in AHPL language. Parity checking has been incorporated at many places and indicator lights provided to warn the user about errors. The user can have access to these error conditions and automatically initiate shut down or any other procedures.

It would have been nice if an existing microprocessor served our purpose. IMP chips were found to be closest to our requirements. However it was concluded that none of the available microprocessor chips could be readily used to implement a decision table processor.

The design can also be integrated into two LSI chips (cf. : sec. 7.3). Such PLC's are expected to have a big market because of their wide applicability and lower cost due to large scale integration.

We may compare our design with some of existing PLC's already in the market. The things to be compared are storage efficiency, scan time, troubleshooting capability and controller's safety.

Storage Efficiency

It is difficult to compare two different methods i.e. decision tables and ladder diagrams objectively. Some indirect method though not reliable may be used. Texas Instruments claim that their 5 TI programmable logic controller system which can have at the maximum 1-K of memory-words, is designed for systems equivalent to 15-200 relays. If we assume that on average, a decision table has 5 conditions, 4 rules and 2 actions then it will require 15 words of memory. Therefore about 275 decision tables can be accomodated into one 4K memory block. These 275 decision tables may be equivalent to about the same number of relays. We see that 5TI controller fares better by a factor of approx. 3 in storage. There are others which do not fare so well as 5 TI.

Scan Time :

Scan time for available PLC's for 4K memory varies from 12 msecs to 40 msecs. Texas Instrument's PLC with maximum of 1K words of memory has scan time of 8.3 msecs and response time of 21 msecs. maximum. If we assume some average behaviour about decision tables like it has 5 conditions (of which three require access to I/O logic block and two require the access to IFP), 4 rules (of which 2 rules are examined), and 2 actions (of which one require access to I/O logic block and other to IFP) then we find a single decision table takes 93 micro-secs and 275 decision tables take 26 msecs (Clock frequency assumed = 2 MHZ). In our case 26 msec. is also the response time. It seems that different manufacturers are assigning different meanings to scantime and so we should be cautious while comparing them with ours. Scan time of 26 msecs is acceptable for control purpose.

Troubleshooting Capability -

We have provided display facilities to the users. This will help one in troubleshooting his program. Similar facilities are provided by the majority of existing PLC manufacturers.

The two indicators e1, e2 provided, locate the fault in I/O logic block and Main Controller respectively. Similar facilities are now being provided in existing PLC's also. But something more need be done to pin point the fault.

Safety of Programmable Logic Controllers

Now the manufacturers have started providing some protection features in their PLC's. The extent of protection is not known to the author. We also , on adhoc basis , have incorporated parity checking at a number of places and have coded input output address into 11 bits code before it is sent to I/O logic block. Selection logic has been duplicated (cf: Chapter VI) to ensure that we receive the correct information from input output logic block.

The operating scan time in case of our PLC is expected to be much lower than 26 msecs. This is because of availability of powerful actions 'GDE' and GDT . It is generally the case that a process to be controlled has three phases viz. start-up, normal, and stop. By the use of GDE and GDT actions, one can confine oneself to those decision tables which represent the current operative phase only and hence it will lead to saving the PLC time. This facility is not available on any of the existing PLC's.

EE-1975-M-PRA-PRO